University of Alberta

SPREADSHEET REVERSE ENGINEERING

by

Dabo Sun

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of **Master of Science**.

Department of Computing Science

Edmonton, Alberta
Spring 2007

Library and Archives Canada

Published Heritage Branch

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque et Archives Canada

Direction du Patrimoine de l'édition

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence
ISBN: 978-0-494-30030-5
Our file Notre référence
ISBN: 978-0-494-30030-5

NOTICE:
The author has granted a non-exclusive license allowing Library and Archives Canada to reproduce, publish, archive, preserve, conserve, communicate to the public by telecommunication or on the Internet, loan, distribute and sell theses worldwide, for commercial or non-commercial purposes, in microform, paper, electronic and/or any other formats.

The author retains copyright ownership and moral rights in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

AVIS:
L'auteur a accordé une licence non exclusive permettant à la Bibliothèque et Archives Canada de reproduire, publier, archiver, sauvegarder, conserver, transmettre au public par télécommunication ou par l'Internet, prêter, distribuer et vendre des thèses partout dans le monde, à des fins commerciales ou autres, sur support microforme, papier, électronique et/ou autres formats.

L'auteur conserve la propriété du droit d'auteur et des droits moraux qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this thesis.

While these forms may be included in the document page count, their removal does not represent any loss of content from the thesis.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de cette thèse.

Bien que ces formulaires aient inclus dans la pagination, il n'y aura aucun contenu manquant.

Canada

To my parents and Xiaolei.

Abstract

Spreadsheet applications are very popular end-user programming environments, and many spreadsheet documents are crucial because they keep records of important information, calculation, and analysis. Researchers have discovered, however, that spreadsheets are often error prone, hard to understand, maintain, and reuse. In this thesis work, reverse engineering and visualization techniques are applied to reveal the hidden dependencies between cells and computation units, as well as high-level structural and conceptual abstractions of spreadsheets. A toolkit is developed to ease end-user understanding of spreadsheets by extracting the artifacts and dependencies, analyzing and visualizing the facts, and detecting anomalies. Various evaluation techniques have been used to assess the usability of the tool. In particular, a user study has been conducted to validate the usability and effectiveness of the tool. Results show that our method is very effective in detecting dependencies hidden in spreadsheets, and the tool is easy to learn and use.

Acknowledgements

First of all, I would like to thank my supervisor, Dr. Ken Wong, for his ideas, advice, guidance, and encouragement throughout my study. This work would not be possible without his support.

I would like to thank Dr. Jim Hoover, as well as other members in the software engineering group at the University of Alberta, who gave me many good suggestions on my work.

I am very grateful to my committee members, Dr. Eleni Stroulia and Dr. Marek Reformat, for their time to read my thesis and valuable feedback on my work.

Many thanks go to Dr. Margaret-Anne Storey, Robert Lintern, and Chris Callendar, from the CHISEL research group at the University of Victoria for their support on SHriMP, a software visualization tool.

I thank all the participants in the user study for their time and effort.

Thanks also go to my friends Marianne Morris, Chunyan Meng, Zhenchang Xing, Stanley Oliveira, Yingtao Jiang, Daqing Hou, Ying Liu, and Daniel Moise for their help in my studies, research and everyday life.

I cannot express my appreciation in words to my parents for their constant love, care, and understanding. Whenever difficulties occur, they are always there to support me.

Last, but definitely not least, I thank my girlfriend, Xiaolei, for her love, support, and encouragement. She makes my life beautiful.

Table of Contents

1 Introduction 1
- 1.1 Motivations . . . 1
- 1.2 Problem . . . 2
- 1.3 Approach . . . 3
- 1.4 Contributions . . . 3
- 1.5 Organization . . . 4

2 Background 5
- 2.1 Spreadsheet concepts . . . 5
- 2.2 Spreadsheet errors . . . 6
- 2.3 Software reverse engineering . . . 8
- 2.4 Program comprehension . . . 10
- 2.5 Information visualization . . . 11

3 Approach and system architecture 13
- 3.1 Requirements . . . 13
  - 3.1.1 Basic requirements . . . 13
  - 3.1.2 Additional requirements . . . 13
  - 3.1.3 Notebook requirements . . . 14
- 3.2 System architecture . . . 15
  - 3.2.1 Excel . . . 16
  - 3.2.2 SHriMP . . . 16
  - 3.2.3 Interactions . . . 17
- 3.3 Spreadsheet Analyzer . . . 18
  - 3.3.1 Reading Excel files . . . 18
  - 3.3.2 Graph eXchange Language (GXL) . . . 19
  - 3.3.3 Extracting artifacts . . . 21
  - 3.3.4 Parsing formulas . . . 22
  - 3.3.5 Building abstractions . . . 23
  - 3.3.6 Detecting anomalies . . . 26
  - 3.3.7 Generating GXL graphs . . . 28
- 3.4 Tool integration . . . 28
  - 3.4.1 Integrating the Spreadsheet Analyzer with SHriMP and Excel . . . 30
  - 3.4.2 Bridging SHriMP-Cell and Excel . . . 30

4 Evaluation 33
- 4.1 Guideline-based evaluation . . . 33
  - 4.1.1 Heuristic evaluation . . . 34
  - 4.1.2 Cognitive dimensions . . . 36
- 4.2 User study . . . 39
  - 4.2.1 Goals . . . 39
  - 4.2.2 Pilot experiments . . . 39
  - 4.2.3 Participants . . . 40
  - 4.2.4 Experiment design . . . 40
  - 4.2.5 Phases . . . 41
  - 4.2.6 Experiment variables . . . 42

4.2.7 Experiment results . . . 43
4.3 Discussion . . . 51
4.4 Theory-based evaluation . . . 52
4.4.1 Theories of perception . . . 52
4.4.2 Perceptual organization . . . 53
4.4.3 Perceptual segregation . . . 55
4.4.4 Applying perceptual theories . . . 56

5 Related work **58**
5.1 Spreadsheet visualization . . . 58
5.2 Spreadsheet auditing . . . 59
5.3 Tool comparisons . . . 59

6 Conclusions and future work **61**
6.1 Lessons learned . . . 61
6.2 Recommendations . . . 62
6.3 Summary . . . 64
6.4 Future work . . . 64

Bibliography **66**

A Solicitation Letter **71**

B Pre-study questionnaire **73**

C Consent form **74**
C.1 Purpose of the study . . . 74
C.2 Procedure . . . 74
C.3 Your rights . . . 74
C.4 Principal investigators . . . 75
C.5 Agreement . . . 75

D Experimenter's handbook for SHriMP-Cell **77**
D.1 General procedures . . . 77
D.1.1 Phases . . . 77
D.1.2 Principles . . . 78
D.2 Orientation . . . 78
D.3 Training . . . 79
D.3.1 Setup . . . 79
D.3.2 Background . . . 79
D.3.3 Key features of SHriMP-Cell . . . 80
D.4 Practice . . . 81
D.4.1 Setup . . . 81
D.4.2 Practice tasks . . . 81
D.5 Formal tasks . . . 82
D.5.1 Setup . . . 82
D.5.2 Formal tasks . . . 82
D.6 Post-study questionnaire . . . 83
D.7 Post-study interview and debriefing . . . 83

E Experimenter's handbook for Excel **84**
E.1 General procedures . . . 84
E.1.1 Phases . . . 84
E.1.2 Principles . . . 85
E.2 Orientation . . . 85
E.3 Training . . . 86
E.3.1 Setup . . . 86
E.3.2 Background . . . 86
E.3.3 Key features of Excel . . . 86

E.4 Practice 87
E.4.1 Setup 87
E.4.2 Practice tasks 87
E.5 Formal tasks 88
E.5.1 Setup 88
E.5.2 Formal tasks 88
E.6 Post-study questionnaire 89
E.7 Post-study interview and debriefing 89

F Post-study questionnaire **90**

G Post-study interview questions **92**

List of Figures

2.1 The VisiCalc user interface 6
2.2 Classification of spreadsheet errors 7
2.3 An example of the formula view and trace results in Excel 8
2.4 High-level abstraction in Rigi 9
2.5 A Hangman program presented in SHriMP 12
2.6 Spreadsheet structure 12

3.1 The system architecture 16
3.2 The architecture and data flow of the Spreadsheet Analyzer 18
3.3 A structural view of the weather spreadsheet 24
3.4 A cell's possible adjacent cells 24
3.5 A structural view of the weather spreadsheet 25
3.6 A customized view of the weather spreadsheet 27
3.7 An anomaly detected in the weather spreadsheet 28
3.8 An example of partial GXL file 29
3.9 A spreadsheet menu extension to SHriMP 30
3.10 The installer program to install/uninstall the Excel extension 31
3.11 Pop up menu from SHriMP node 31
3.12 Add a cell block using the Excel extension 32

4.1 The evaluation framework in this thesis 34
4.2 Two sets of experiments 41
4.3 Results from the Query View tool for Task 2 45
4.4 Results from the trace dependents tool in Task 2 47
4.5 Arrow directions in Excel and SHriMP-Cell 47
4.6 Results from the Query View tool for Task 4 48
4.7 Results from the trace precedent tool in Task 4 49
4.8 Examples of the law of similarity. 54
4.9 Examples of the law of proximity. 54
4.10 An example of the law of connectedness. 55
4.11 An example of the law of orientation. 55
4.12 User D's answer to Task 4 57

6.1 An example SHriMP view with column-based layout 63

List of Tables

4.1 Formal tasks correctness results 43
4.2 Formal tasks time spent 43
4.3 Post-study questionnaire results 50

Chapter 1

Introduction

1.1 Motivations

Spreadsheet applications are very most popular end-user programming environments, and many spreadsheet documents are crucial because they keep records of important information, calculation, and analysis. Spreadsheet users are considered as end-user programmers. When they create spreadsheets, especially when creating formulas, they are writing software programs. Since computers are widely used, and end-user tools such as office suites and web development tools have become very powerful, the number of end-user programmers has substantially increased. It is estimated that, in 2005 in the United States alone, there are about 55 million end-user programmers, which constitutes nearly one sixth of the US population [21]. It is also observed that a large portion of these end-user programmers are spreadsheet users [60]. However, researchers have discovered that spreadsheets are often error prone, hard to understand, maintain, and reuse. Furthermore, it is often difficult for the original spreadsheet authors to document the design of a spreadsheet, and explain it to others [38]. The reality is that many business people and politicians make critical decisions depending on data gathered and calculated by spreadsheet documents, even when they contain wrong numbers. Consequently, millions of dollars are lost each year due to spreadsheet errors.

End-user programmers, who write far more software than professional programmers, usually do not have professional knowledge and training in information technology. They rely much on end-user tools and their domain knowledge. For example, spreadsheet users usually do not have much knowledge of programming in a procedural or object-oriented programming language, but they know what needs to be computed and presented. They can simply input numbers, create formulas, and compute the results, since spreadsheet documents merge the computation model with the presentation. Also, traditional software engi-

neering approaches on quality control do not apply well to the end-user community. As a result, about 40% to 50% of software created by end users contains non-trivial errors [60]. How to better support end-user programmers to reduce errors and improve quality has become an emerging research area.

1.2 Problem

There are many reasons why spreadsheet documents are so error-prone and hard to maintain, such as the lack of professional training and "programming" conventions for spreadsheet users, the complexity of the problem domain, and the limitations of spreadsheet applications. Most importantly, many researchers point out that the major problem of spreadsheets is their hidden structure; i.e., the computation layer is often hidden under the presentation layer [26, 69]. The invisibility of the structure of a computation adds great difficulties to spreadsheet users to understand and audit spreadsheets. Moreover, it is not easy to observe the computational data flow and the high-level structures of spreadsheet documents, because the actual data dependencies are often too dense to see at-a-glance overlaid on the tabular view.

In the early stage of this thesis work, we informally interviewed a few spreadsheet users and asked them about the problems they had with spreadsheet applications and documents. One user told her story. Once, she went on vacation for a while. When she came back to work, she was very upset to find that some of her report spreadsheets were totally destroyed. Many formulas were removed and replaced by hard-coded numbers. She was told that the company needed an updated report for a conference, but no one could completely understand how her spreadsheets worked. They tried to change some numbers, but the results were not what they expected. So, they had to remove some formulas in order to create a "correct" report and keep the same presentation style.

This thesis addresses the following questions:

- What is an effective way to reveal a spreadsheet's hidden structures?
- What tool support can we provide for end users to ease spreadsheet understanding?
- Is such a tool effective and easy to use?

1.3 Approach

This thesis applies reverse engineering and visualization techniques to reveal the hidden dependencies between cells and computation units (a block of cells), as well as the high-level structural and conceptual models of a spreadsheet document. We believe that reverse engineering can help users to understand spreadsheet structures, potentially reduce errors, and improve quality. A toolset is developed to extract spreadsheet artifacts, analyze and visualize spreadsheets, and detect anomalies.

Reverse engineering techniques are used to extract low-level facts from cells and their dependencies. Based on the fact extraction, spreadsheet artifacts are analyzed and visualized using nested graphs to provide alternative views of high-level and low-level structures. Furthermore, anomalies (i.e., areas where errors might occur) are detected and highlighted. The major steps are listed below.

- A Spreadsheet Analyzer reads the spreadsheet document and extracts low-level facts, such as cells, formulas, comments, and cell referencing information.
- The Spreadsheet Analyzer automatically clusters the low level artifacts into higher-level blocks based on the spatial layout and the computational data flow.
- Anomalous formulas can be identified by the Spreadsheet Analyzer.
- Using an Excel add-on tool, users can cluster cells based on their understanding of the spreadsheet.
- GXL graphs are generated based on the low-level artifacts and the high-level abstractions (clusters).
- SHriMP, an information visualizer, is integrated with the Spreadsheet Analyzer to visualize spreadsheets in different views, and present direct and indirect cell dependencies.

1.4 Contributions

The main contributions of this thesis on reverse engineering spreadsheets are:

1. A reverse engineering methodology has been proposed to improve the understandability of spreadsheet documents.

2. A platform-independent Spreadsheet Analyzer was developed to extract low-level artifacts from spreadsheet documents, build high-level abstractions, and detect anomalies.

3. A visualizer was extended and integrated with the Spreadsheet Analyzer to visualize spreadsheet documents using multiple views to support various comprehension needs.

4. Various evaluation techniques have been used to evaluate the usability of our toolset. In particular, a user study has been conducted to evaluate the usability and effectiveness of the toolset.

1.5 Organization

The rest of this thesis is organized as follows. Chapter 2 provides an overview of the background knowledge to understand this thesis. Chapter 3 introduces the proposed method and design of the toolset. Chapter 4 evaluates the proposed method and the toolset. Chapter 5 reviews the related work, and compares our toolset with some other tools. Finally, Chapter 6 draws conclusions, summarizes lessons learned, and proposes future work.

Chapter 2

Background

2.1 Spreadsheet concepts

A typical spreadsheet document looks just like a table, which contains many cells organized into rows and columns. A *cell* has a coordinate (also called a cell reference), which is usually labeled by its column (a letter) and its row (a number) position in the table. For example, "A1" represents the cell in the first column and first row. A non-empty cell contains a value, which is either a literal, or a result computed from a *formula*. A cell can refer to other cells by using a formula. Formulas provide powerful computation capabilities to spreadsheet documents and distinguishes them from normal data tables. A cell may also have a comment attached, which often contains important information about the cell. Many modern electronic spreadsheet applications have the ability to visualize tabular data in various kinds of charts such as bar charts, line charts, and pie charts, etc. Some spreadsheet applications even allow users to add multi-media objects such as pictures and videos. Since the focus of this thesis work is on the structure of spreadsheet documents, i.e., how cells are related to each other, these presentation-oriented objects are not considered for analysis and visualization.

The first electronic spreadsheet application was VisiCalc, designed and developed by Dan Bricklin and Bob Frankston in 1979 [23]. The original idea of VisiCalc was to extend a calculator so that it can be more effective for their business school projects [45]. Later, VisiCalc evolved into a fully functioning spreadsheet application, and became very popular in industry. Figure 2.1 shows the main user interface of VisiCalc. The tabular view is very similar to modern spreadsheet applications. VisiCalc uses the "A1" format for cell coordinates.

Since VisiCalc was invented, many software companies have released their electronic spreadsheet applications. Although different spreadsheet applications may provide different

Figure 2.1: The VisiCalc user interface

features, core concepts such as the cell-formula model remain. Examples of popular spreadsheet applications are Microsoft Office Excel [49], Lotus 123 [40], OpenOffice Calc [6], Gnumeric [3], etc. Excel, a component of Microsoft Office Suite, is probably the most popular spreadsheet application because of the popularity of Microsoft Office. Therefore, Excel spreadsheet documents were investigated in this thesis.

One of the unique features of spreadsheet documents is that computation and presentation are combined together, and a user can see what has been done incrementally. In contradiction, professional software engineers usually try to separate computation and presentation in program design to improve maintainability and reusability; e.g., Model-View-Controller (MVC) is one of the most often-used design patterns [30]. Although the mix of computation and presentation may increase comprehension and maintenance difficulties, end users like the convenience of What You See Is What You Get (WYSIWYG), especially if they do not have any programming skills or professional training.

Spreadsheet applications are widely used in many domains such as finance, science, and engineering. In the finance world, spreadsheet documents have become the "de facto language of business" [46]. In science and engineering, spreadsheet documents are used in many fields for modeling, statistics, and equation computation [19, 20]. From here on, we use the term "spreadsheet" to refer to spreadsheet document, not spreadsheet application.

2.2 Spreadsheet errors

Spreadsheet error studies are the foundation of spreadsheet research, as errors not only reveal the problems people have in using spreadsheet applications and documents, but also indicate the limitations of spreadsheets themselves.

Panko did some pioneering work on spreadsheet errors. He estimated that as the size of a spreadsheet increases, its error rate may rise from 20almost 90%, which means that for very large spreadsheets, the question is not whether errors exist, but rather how many errors there are. He observed that errors often occur in a small percentage of cells [60, 61]. Panko also found that despite the high error rate in large spreadsheets, end users are still very confident about their work and believe that they can create error-free spreadsheets.

Rajalingham and his colleagues proposed a framework to classify of spreadsheet errors as shown in Figure 2.2 [65]. The classification reveals the nature of spreadsheet errors, and it is useful to study specific types of errors, and design tools and techniques to prevent certain types of errors.

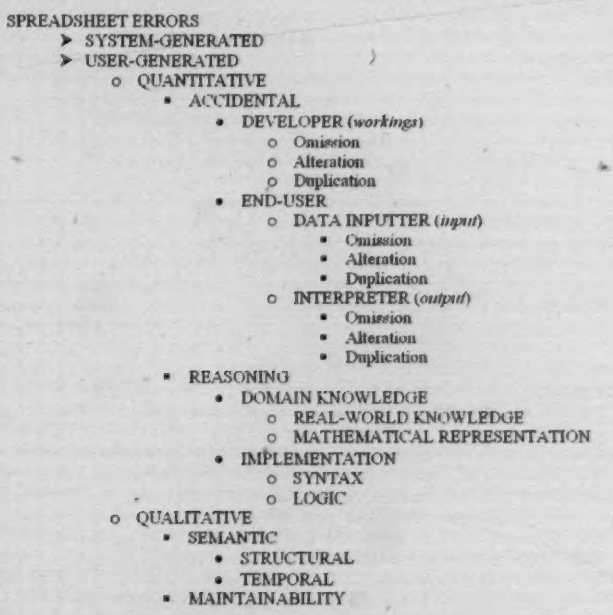


Figure 2.2: Classification of spreadsheet errors

Tool support has been designed to audit spreadsheet errors. For example, Microsoft Excel has a formula auditing mode (also called "formula view"), which displays the content of all formulas if any, instead of the resulting values. Excel also has a built-in trace tool (called "formula evaluator"), which allows users to trace a selected cell's precedents (cells that the selected cell refers to) and dependents (cells that refer to the selected cell). Figure 2.3 shows the formula view and trace results.

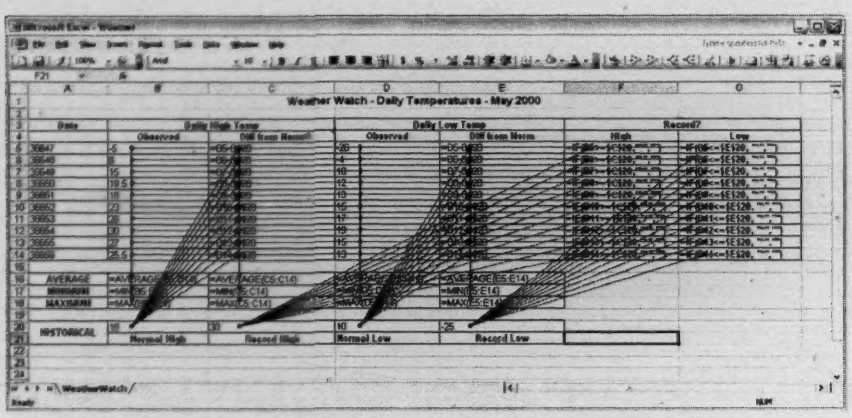


Figure 2.3: An example of the formula view and trace results in Excel

2.3 Software reverse engineering

Reverse engineering techniques were originally used in the hardware industry to decipher designs from finished products. In the software engineering community, reverse engineering is defined as analyzing a software system to identity its subsystems, components, and their relationships [28].

A typical process to reverse engineer a software system involves fact extraction, analysis, and visualization [50, 85].

- **Fact extraction**: The source code (e.g. Java, C/C++) is transformed into another format, which can be further analyzed or loaded into a visualizer. At the fact extraction stage, various software artifacts such as files, classes, methods, etc., are extracted and stored depending on the nature of the subject system and user need.

- **Analysis**: The raw graph of a software system is often huge, and it needs to be decomposed and clustered into higher levels of abstraction to be more understandable. Common tasks in the analysis stage include system decomposition, metrics calculation, and structure recovery.

- **Visualization**: To make the software artifacts and their relationships readable and understandable, they can be visualized, often in a graph presentation. The graphs not only present various entities and attributes, but also highlight relationships so users can easily tell which components are related to a specific artifact.

Figure 2.4 shows a reverse engineered system in Rigi, a visual tool for understanding legacy systems [7]. The graph on the left-hand side shows all the artifacts extracted from

the subject system. A huge graph with millions of nodes and arcs is not very helpful, and it only highlights the complexity of the subject system. Through system analysis and decomposition, a high-level abstraction of the system can be created which represents the user's mental model of the subject system, as shown on the right-hand side.

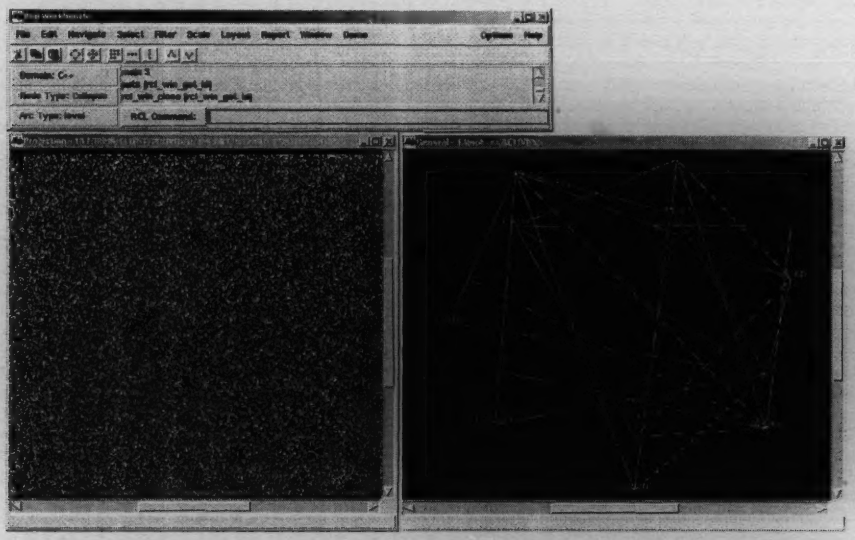


Figure 2.4: High-level abstraction in Rigi

Reverse engineering does not have to be applied only to finished systems. It can and should be used at any phase of the software life cycle to improve quality, reusability, and maintainability. Wong proposed a Reverse Engineering Notebook approach to continuously apply reverse engineering in various stages of a software system [83]. Unfortunately, reverse engineering is often not widely accepted because software engineering has focused more on software construction than software maintenance and evolution [55]. Also, reverse engineering is usually hard to achieve, and not everything can be reverse engineered from source code; some design ideas might only be in the original designer's mind. Unlike forward engineering, which has strong theoretical foundations (e.g., programming languages, compilers, etc.) and various techniques such as data structures, design patterns, application frameworks, and software product lines, software reverse engineering is usually performed in an ad-hoc manner and heavily depends on a reverse engineer's experience [54].

2.4 Program comprehension

The main purpose of software reverse engineering is to help people understand a software system to ease maintenance and reuse activities. Such systems are often old, large, and poorly documented. Source code becomes the most reliable resource [54]. Thus, one of the main tasks of program comprehension is to develop mental models of the current state of a system from source code.

It is observed that people often use different methods to understand a software system, and various comprehension models have been proposed [73]:

- **Bottom-up**: understanding by reading source code, and then mentally grouping the source code statements into higher-level abstractions.
- **Top-down**: applying higher-level domain knowledge into understanding source code, driven by hypotheses.
- **Knowledge-based**: assimilating a knowledge base (programmer's expertise and background knowledge) and mental model (programmer's current understanding of the program).
- **Systematic and as-needed**: reading the code in detail or only focusing on code related to a specific task.
- **Integrated approaches**: switching between different models depending on the task since program comprehension is usually not an end goal, but rather a prerequisite to achieve some other objective [72].

Most of the studies done to create these comprehension models are based on professional programmers and software engineers. However, the question is whether end-user programmers have the same comprehension models. Hendry and Green did an empirical study on how spreadsheet users create, comprehend, and explain spreadsheets. The results show that spreadsheet users use similar strategies to understand spreadsheets as professional programmers do in other software systems [38]:

- **Bottom-up**: end users might start from a formula, trace all the dependent cells of the formula to understand how the calculation works.
- **Top-down**: end users might read the title, labels, and comments of each worksheet to understand what the spreadsheet is about.

- **Knowledge-based**: end users might try to use their domain knowledge to understand a spreadsheet. For example, when they see the term "income tax" in a spreadsheet, they will use their knowledge of the tax system to understand the spreadsheet.
- **Integrated approaches**: end users might switch between various strategies to understand a spreadsheet.

This interesting study on spreadsheet "programmers" indicates that reverse engineering techniques and program comprehension tools designed for professional programmers could be applied to end-user programmers, since they have similar ways to understand a system as professional programmers do.

2.5 Information visualization

Information visualization can be defined as "the use of interactive, sensory representations, typically visual, of abstract data to reinforce cognition" [10]. The visual representation of data may include graphs or animations. An example of information visualization is a city map, which transforms and represents complicated geographical and social information on a two-dimensional diagram to help users better understand and retrieve data. According to Ware, the advantages of visualization include the following aspects [82].

- Visualization is capable of summarizing huge amounts of data.
- "Visualization allows the perception of emergent properties that were not anticipated".
- Visualization can often make non-trivial problems apparent. This is very useful for error detection; with an appropriate visualization, errors can be revealed easily.
- Visualization can help to understand both large-scale and small-scale features of data.

Information visualization can be applied to many domains as presented in [35]. In software engineering, visualization is often used to explore software structure and navigate source code [74]. SHriMP (Simple Hierarchical Multi-Perspective) is a software visualization environment and technique to visualize and explore software systems in multiple perspectives [8, 74]. Figure 2.5 shows how SHriMP can present a Hangman program written in C.

The SHriMP visualization technique uses nested graphs to present the hierarchical structure of a software system and its subsystems. The technique effectively avoids multiple

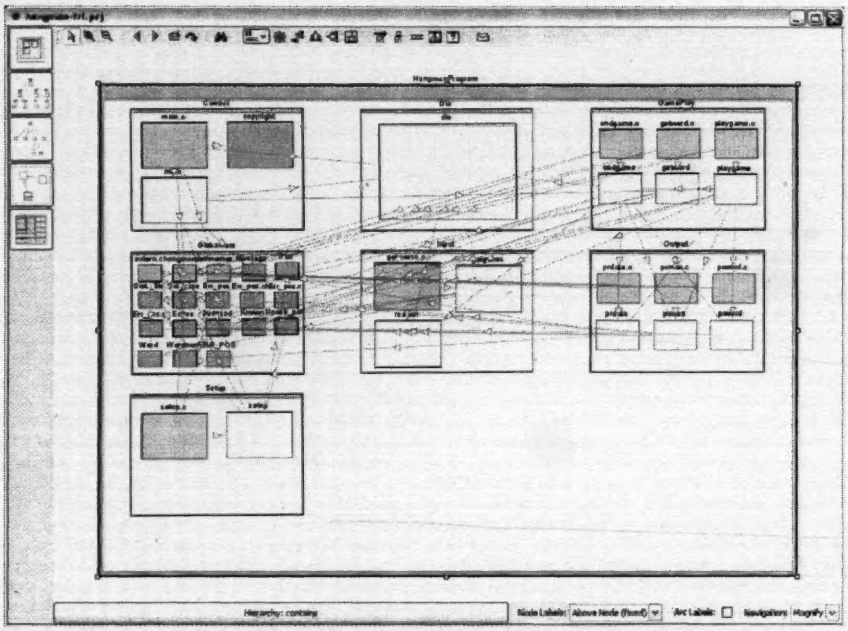


Figure 2.5: A Hangman program presented in SHriMP

windows, which could be disorienting. Spreadsheet structures can be viewed as a hierarchy where the top level is a workbook which contains one or more worksheets. In each worksheet, there might be many ranges (or blocks) which could be physical, computational, or conceptual groupings, and each range has low-level cells. This spreadsheet structure is shown in Figure 2.6. Therefore, SHriMP views can be applied to visualize the hierarchical structure of a spreadsheet.

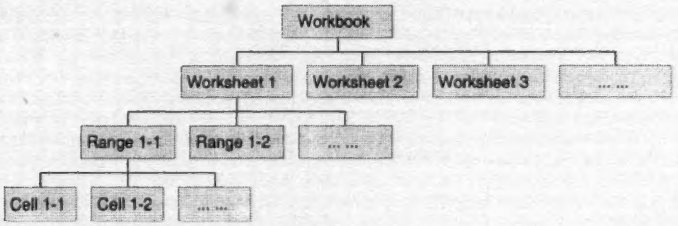


Figure 2.6: Spreadsheet structure

Chapter 3

Approach and system architecture

This chapter describes the requirements, architecture, and process of the developed toolset that parses, analyzes, and visualizes spreadsheets in both low-level artifacts and high-level abstractions.

3.1 Requirements

3.1.1 Basic requirements

To implement the spreadsheet reverse engineering approach, the toolset should support three key requirements [79]:

- **Extraction**: The toolset should be able to read a spreadsheet file and extract various artifacts such as worksheets, cells, and dependencies between cells.
- **Abstraction**: Spatial and computational relationships between cells and ranges should be identified to build higher-level abstractions to help users build a mental model of a spreadsheet.
- **Visualization**: The toolset should provide alternative views for both low-level cells and high-level abstractions to aid comprehension.

3.1.2 Additional requirements

Furthermore, based on our observation of spreadsheet users (by literature review, interviewing spreadsheet users based on an early toolset prototype, and studying the questions on an Excel newsgroup [48]) and our experience on building software tools, we propose the following requirements to improve usability and adoptability.

- **Interaction**: Visualization techniques can provide the user alternative views of spreadsheets, perhaps graph-based. However, the user might also get lost in nodes and arcs, and their correspondence with the original cell columns and rows. Thus, integration between the spreadsheet application and the visualization tool is required to avoid disorientation. Nevertheless, we find that many spreadsheet visualization tools do not support interactions between a graph view and the original spreadsheet tabular view well [14].

- **Persistence**: Artifacts and abstraction information should be stored persistently, and be easily retrievable, since they contain both the physical and conceptual structure of spreadsheets. This information, over time, also reveals the evolution of a spreadsheet, as well as the users' past mental models, which could be useful to analyze for research and educational purposes. Thus, an efficient way to store and retrieve this information is required.

- **Extensibility (or interoperability)**: The toolset should be extensible to improve adoption. For example, the toolset should be easily extended to support another visualization interface, such as a web-based interface.

3.1.3 Notebook requirements

Wong proposed a continuous program understanding notebook infrastructure, which has been referred by many reverse engineering tool builders as a design framework [83]. We briefly preview how some of the Notebook requirements are addressed.

> *Requirement 2 (Program Understanding): "Recognize that design abstractions are useful not only for the initial design but also as a way to convey high-level understanding during evolution".*

Structural and computational views can be used throughout the evolution of a spreadsheet, as they are always consistent with the latest version of the document.

> *Requirement 6 (Reverse Engineering): "Produce useful summaries of software structure while highlighting anomalies and filtering irrelevant details".*

Anomalies can be detected by the Spreadsheet Analyzer and visualized by SHriMP. A computational view filters out all the non-computational nodes. At any time, users can show or hide specific types of nodes (representing cells or blocks) and arcs (representing dependencies).

Requirement 11 (User Needs): "Support the continuous understanding of software".

SHriMP-Cell can be used at any time, and abstraction graphs can be persisted for further analysis.

Requirement 15 (Cognitive Issues): "Provide interactive, consistent, and integrated views, with the user in control".

Interaction has been implemented to allow the user to take control. Also, users can group cells in the customized view.

Requirement 17 (Process): "Organize the diverse array of information artifacts involved in software understanding".

Low-level artifacts, high-level abstractions, and their relationships are organized in GXL graphs.

Requirement 18 (Documentation): "Provide consistent, continually updated, hypermedia software documentation".

Users can zoom into any cell node to check the details of an artifact. When spreadsheets change, the toolset can reload updated GXL graphs.

Requirement 20 (Human-computer interaction): "Integrate graphical and textual software views, where effective and appropriate".

SHriMP views have been integrated with the spreadsheet view to ease comprehension.

3.2 System architecture

Figure 3.1 shows the overall architecture of the toolset. There are three main components: Excel, SHriMP, and Spreadsheet Analyzer. SHriMP provides graph views based on the artifacts and abstractions extracted from a spreadsheet. SHriMP views and Excel tabular views together serve as the front end. The analyzer, which serves as the back end, reads and parses Excel files, extracts useful information, and builds higher-level abstractions.

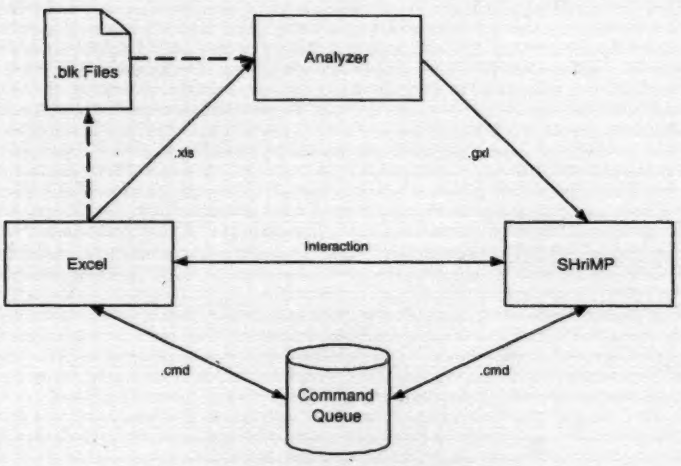


Figure 3.1: The system architecture

3.2.1 Excel

Excel is the spreadsheet application used to read and edit spreadsheets. Excel has powerful calculation capabilities, rich graphical tools, and a scripting interface. Visual Basic for Applications (VBA) allows programmers to add functionality. An extension of Excel is developed in VBA as part of the toolset, which supports interactions between Excel and SHriMP, and offers user assistance in building high-level abstractions. This Excel extension will be described in Section 3.4.2.

3.2.2 SHriMP

SHriMP is a domain-independent information visualization tool designed to help people navigate complex information spaces and hierarchical structures. SHriMP visualizes an information space in one nested graph, and users can zoom into any level of abstraction. This avoids multiple windows, which can be disorienting to users. SHriMP has been integrated with the Spreadsheet Analyzer to provide both high-level and low-level views of spreadsheets. SHriMP is adopted because it supports various comprehension strategies introduced in Section 2.4. SHriMP is well designed and implemented using a Java Beans architecture [78], which is easy to extend and reuse.

In SHriMP, various data input formats such as GXL (Graph eXchange Language) [39], RSF (Rigi Standard Format) [7], XML (Extensible Markup Language) [2], and XMI (XML Metadata Interchange) [11] are supported. GXL is chosen as the input data format to represent spreadsheets, because it is widely used as a graph format in reverse engineering tools.

It is also an XML-based language, so it is easy to store and retrieve. We have explored an efficient way to classify XML-based data so that they can be stored and retrieved efficiently [64].

3.2.3 Interactions

As mentioned before, interactions between SHriMP and Excel are important, but the communication is not easy to achieve because SHriMP is implemented in Java, and Java cannot "talk" to VBA directly. To address this issue, we considered three design solutions.

1. Implement a message "server", which allows SHriMP and Excel to communicate by sending messages (a set of commands). The "server" can be implemented in different ways:
   - Simply create a "command queue file" to store commands for each tool. SHriMP and Excel can communicate by reading and writing the appropriate file.
   - Using a database to store "commands". SHriMP and Excel can communicate by accessing the database.
2. Using "Com-bridge" middleware such as the commercial tool J-Integra for COM [4] or an open source tool Jacob [9] to fill the gap between Java and Excel.
3. Reengineer SHriMP in Microsoft Visual J++ (VJ) or J# so that it can interact with Excel by using VJ or J#'s functions [5].

The last two solutions are discarded because they have to be implemented on Microsoft Windows. Our design philosophy is to increase the adoptability of the toolset. Therefore, the toolset itself should not heavily depend on a certain operating system. The "command queue" solution is adopted, and we choose to use files instead of a database since the toolset currently only supports single-user interaction (i.e., there are no command data consistency issues). As shown in Figure 3.1, a ".cmd" file will be generated for each Excel file once it is loaded into the toolset. When the user selects a cell node in SHriMP and chooses "Show in Excel", a command will be pushed into the command queue for Excel. Then, the user can switch to Excel and click on the "Do SHriMP Request" icon. The corresponding cell will be highlighted in Excel so that the user can get a mapping between SHriMP and Excel.

3.3 Spreadsheet Analyzer

The core component of the toolset is the Spreadsheet Analyzer, which can be seen as a "bridge" between SHriMP and Excel. The Spreadsheet Analyzer reads and parses Excel files into Java objects, identifies cell types and cell relationships, builds high-level abstractions, and generates GXL graphs, which can be visualized in SHriMP. Figure 3.2 shows the modules and data flow of the Spreadsheet Analyzer. There are four modules in the analyzer: Excel Reader, Fact Extractor, Abstraction Builder, and GXL Generator.

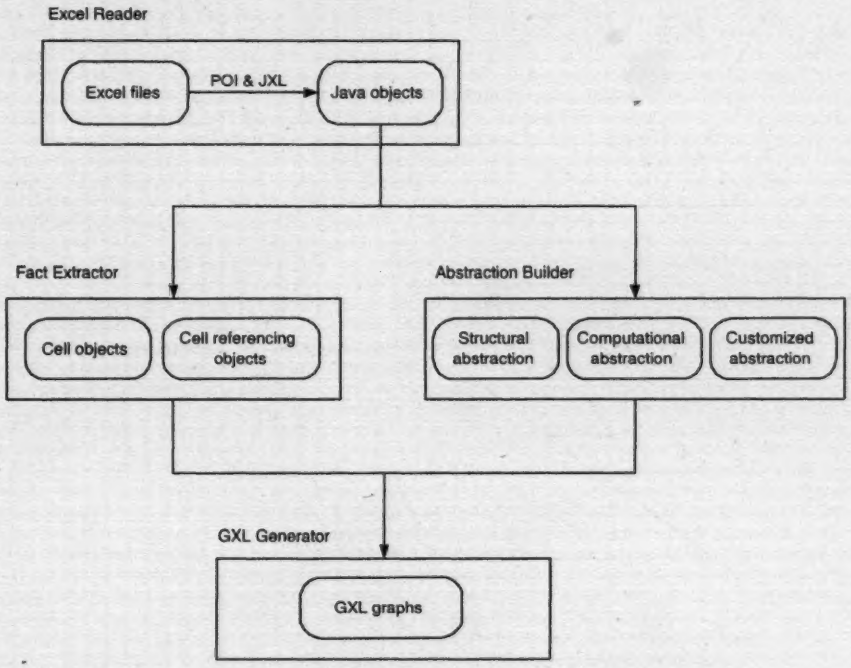


Figure 3.2: The architecture and data flow of the Spreadsheet Analyzer

3.3.1 Reading Excel files

Excel uses a type of Binary Interchange File Format (BIFF) to store spreadsheets. Different versions of Excel use different versions of BIFF. For example, Excel 10.0 (Excel XP) and 11.0 (Excel 2003) use BIFF8X, while Excel 8.0 (Excel 97) and 9.0 (Excel 2000) use BIFF8. Rentz wrote a very detailed document on the Excel file format as part of the OpenOffice project [66]. Application Programming Interfaces (APIs) have been developed to support reading Excel documents in various languages, such as C, Java, Perl, Tcl, and PHP.

Two Java APIs are adopted to read Excel files: Apache Jakarta POI [1] and Java Excel

API (JXL) [43]. POI (Poor Obfuscation Implementation) provides APIs to manipulate various Microsoft office file formats such as Word and Excel. HSSF (Horrible Spreadsheet Format) is POI's subproject to read and write Excel files using Java. Similar to POI-HSSF, JXL is another Java API to read, write, and modify Excel files. After testing both APIs, we found that neither of these two APIs is "perfect" to fit our needs. POI-HSSF is more powerful dealing with Excel formatting information such as frame borders, outlines, etc., but has limited formula support. JXL supports almost all Excel built-in functions, but cannot handle some formatting information, such as outlines, which might be useful for future analysis. POI also has a "shared formula" problem: if a spreadsheet has formulas that are copied from other formulas, Excel will create a shared formula to represent these formulas, and POI will fail to read them. The only solution we could find is to somehow "reenter" these formulas (i.e., copy each formula string and paste it into each formula cell one by one). However, every time the spreadsheet is changed, and saved again, these formula strings need to be retyped again. This problem made scalability testing impossible, and we hope the POI development team can fix this problem in the next release. Therefore, both POI and JXL APIs are used to read Excel files to fully extract the low-level facts in this thesis work.

After the original Excel file is read into Java objects, various artifacts can be extracted. We are interested in the following kinds of artifacts.

- **Cells**: A cell could be a data cell (also called a literal, such as a number or string), or a formula (which can refer to other cells).
- **Cell referencing**: Cell referencing happens in formula cells. There are two kinds of cell references: relative reference (e.g., "A1") and absolute reference (e.g., "$A1", "$A$1"). These references should be represented separately because spreadsheet users often make mistakes when dealing with these two kinds of cell references [38].
- **Comments**: A comment is usually a piece of text attached to a cell. Since we observe that comments usually contain important information, we treat them as a special kind of cell and cell reference.

3.3.2 Graph eXchange Language (GXL)

GXL (Graph eXchange Language), an XML sublanguage, has been developed as a standard exchange format for graph-based data to support interoperability between different software engineering tools. GXL is able to represent typed, attributed, directed, ordered

graphs, and it supports hypergraphs (graphs with n-ary edges) and hierarchical graphs (graphs which contain subgraphs). GXL graphs are generated based on the spreadsheet artifacts for SHriMP to then visualize. The following kinds of GXL nodes and edges can be created in the fact extraction stage.

Data cell node: contains a cell's address (reference represented in the "A1" style) and value. For example, for cell B5 with a cell value "-5.0" on the "WeatherWatch" worksheet, the GXL node looks like:

```
<node id="WeatherWatch!B5">
    <type xlink:href="spreadsheet-schema.gxl#Data"/>
    <attr name="value">
        <string>-5.0</string>
    </attr>
</node
```

Formula cell node: contains a cell's address, value, and the formula string. For example, the following GXL node represents cell C5 in the "WeatherWatch" worksheet. Its value is "-23.0", and its formula string is "B5-B$20".

```
<node id="WeatherWatch!C5">
    <type xlink:href="spreadsheet-schema.gxl#Formula"/>
    <attr name="value">
        <string>-23.0</string>
    </attr>
    <attr name="formula">
        <string>B5-B$20</string>
    </attr>
</node>
```

Comment node: contains the original cell's address, and the comment content. For example, the following GXL node represents cell C4's comment.

```
<node id="WeatherWatch!C4!Comment">
    <type xlink:href="spreadsheet-schema.gxl#Comment"/>
    <attr name="value">
        <string>Column C:
        The difference between the observed high temperature
        for the day and the historical normal (average) high
        temperature for this date.
        ... omitted ...
        </string>
    </attr>
</node>
```

Absolute cell reference edge: represents a formula's reference of another cell by absolute cell referencing. For example, the following GXL edge represents that cell C5 refers to (depends on) B20 using absolute referencing.

```
<edge to="WeatherWatch!B20" from="WeatherWatch!C5">
    <type xlink:href="spreadsheet-schema.gxl#Absolute-reference"/>
</edge>
```

Relative cell reference edge: represents a formula's reference of another cell by relative cell referencing. For example, the following GXL edge represents that cell C5 refers to B5 using relative referencing.

```
<edge to="WeatherWatch!B5" from="WeatherWatch!C5">
    <type xlink:href="spreadsheet-schema.gxl#Relative-reference"/>
</edge>
```

Comment reference edge: represents the relationship between a cell and its comment. Note, comments are not cells in spreadsheets, but they often contain important information. Therefore, comments should be captured as well. For example, the following GXL edge ties cell C4 to its comment.

```
<edge to="WeatherWatch!C4!Comment" from="WeatherWatch!C4">
    <type xlink:href="spreadsheet-schema.gxl#Comment-reference"/>
</edge>
```

3.3.3 Extracting artifacts

"Interesting" facts need to be extracted from spreadsheets, and stored in a format that can be further analyzed by the Spreadsheet Analyzer. From the implementation perspective, the basic goal of fact extraction is to build a list of GXL nodes and a list of GXL edges. GXL nodes contain various types of Excel cells such as data cells, formula cells, and comments. GXL edges contain inter-cell relationships such as relative reference, absolute cell reference, and comment reference. Cells are parsed one by one using both POI and JXL. For each cell, there are the following possibilities based on the JXL cell object types.

1. **Empty cell**: if a cell is empty, this cell will be ignored.

2. **Comment cell**: if a cell has a comment, a GXL node will be created based on the current cell address and the content of the comment. A GXL edge will also be created which has a "comment reference" type, and points for the current cell node to the comment node.

3. **Formula cell**: if a cell contains a formula, this cell needs to be further analyzed. JXL can identify four different types of formulas based on the resulting value: boolean, date, number, and string. If the formula does not refer to other cells, only one GXL node needs to be created, and no GXL edge needs to be created. If the formula refers

to other cells, not only a GXL node needs to be created from the current cell, but also one or more GXL edges need to be created pointing the current cell to all the cells referred to by the formula. Parsing these formulas will be explained in Section 3.3.4.

4. **Data cell**: if the current cell is non-empty and it does not contain a formula, it is a data cell (could be a label or a number). Only a GXL node needs to be created for a data cell.

At the fact extraction stage, the physical structure of a spreadsheet is revealed, i.e., cell-level artifacts and data dependencies from formulas are identified.

3.3.4 Parsing formulas

If a formula cell refers to other cells, all these dependencies need to be discovered. JXL's formula parser is used to find all the referred cell references in a formula. The class *jxl.biff.formula.StringFormulaParser* can parse a formula string into a parse tree, and list all the tokens such as cell references, operators, function names, etc. Since it is much easier to grab a formula cell's formula string in POI than in JXL, POI's method *getCellFormula()* in class *org.apache.poi.hssf.usermodel.HSSFCell* is used to retrieve a formula string. Then, the formula string can be passed to the *StringFormulaParser* class, which has been extended to get a list of the parsed tokens. From the inter-cell dependency perspective, it does not matter what functions or operators are used. Only the cell references that the current formula cell refers to are important. Therefore, any tokens that are not cell reference or range reference tokens are filtered out by Java regular expressions.

In an Excel worksheet, the maximum number of columns is 256, and the maximum number of rows is 65536. Thus, based on the "A1" cell reference style, a cell reference could start from letter A to IV, and end with a number from 1 to 65536. A dollar sign ($) could appear before either a column or a row to specify absolute cell referencing. The following Java regular expression is used to detect cell references and functions.

```
\\$?([A-Z]|[A-H][A-Z]|[I][A-V])\\$?\\d+
```

For succinctness, the regular expression does not check the row boundary 65536. For example, *IF* is an Excel built-in function, while *IF2* is a cell reference. To match a range reference (e.g., *A1:B5*), two cell reference regular expressions can be concatenated with a colon (:). For range references, every non-empty cell within each range needs to be considered as an individual reference.

3.3.5 Building abstractions

In the fact extraction stage, the physical artifacts such as cells, formulas, comments, and their relationships are identified and stored in lists of GXL node and edge Java objects. Based on these physical artifacts, a spreadsheet can be analyzed to build higher-level abstractions. We define three kinds of abstractions.

1. **Structural abstraction**: structure is based on the spreadsheet's spatial layout. Cells close to each other will be grouped as a block.

2. **Computational abstraction**: identify the input, calculation, and result blocks in each worksheet. Calculation-based spreadsheets (documents that contain formulas) usually have a visually hidden data-flow. Input data will be computed upon to generate results.

3. **Customized abstraction**: users can group cells into blocks arbitrarily to build "conceptual" abstractions. In each problem domain, users usually have certain types of cells based on their domain knowledge. For example, in a finance report spreadsheet, there might be conceptual types such as income, balance, interest, etc. Grouping cells based on semantics would help users to better understand their spreadsheets.

Structural abstraction

Structural abstraction focuses on the spatial layout of a spreadsheet. We observe that spreadsheet users usually follow certain conventions to design their spreadsheet layout so that spatial patterns imply design semantics [36, 56]. Figure 3.3 shows a structural view of the weather spreadsheet. Green rectangles represent data cells, green ellipses represent comments, and green triangles represent formulas. Solid lines represent relative references, and dotted lines represent absolute reference. Users can easily tell the types of cells and cell references at a glance.

The basic idea of the structural abstraction is to partition all non-empty cells into clusters based on the distance between two cells. Cells that are "close" to each other are partitioned into the same cluster. Euclidean distance is used to compute the distance between two cells [34]. The distance between *Cell i* and *Cell j* is defined as following:

$$d(Cell_i, Cell_j) = \sqrt{(Column_i - Column_j)^2 + (Row_i - Row_j)^2}$$

A cluster can be defined as a group of cells, in which the distance between each pair of adjacent cells is equal to or less than a threshold k. Observation of spreadsheets shows

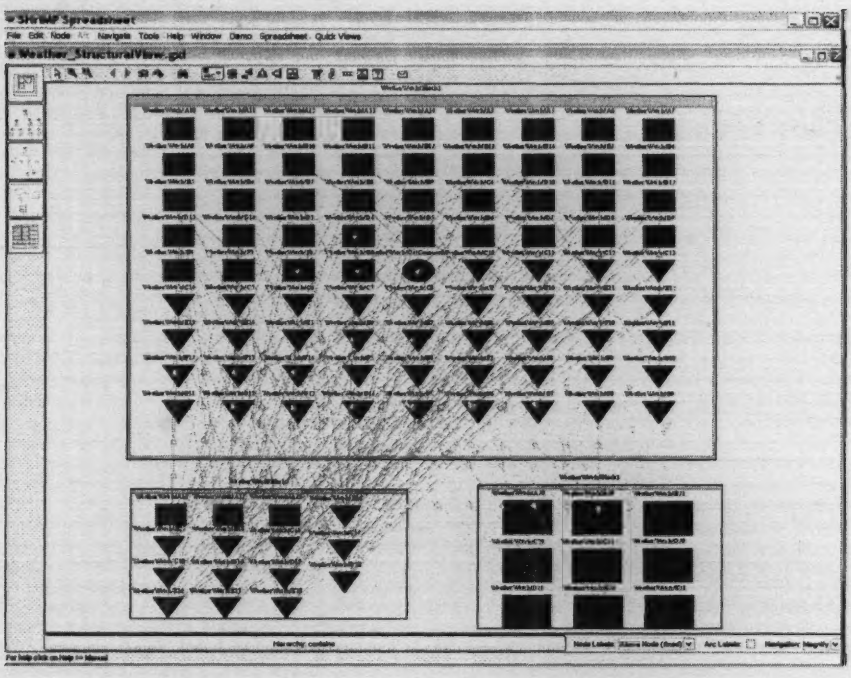


Figure 3.3: A structural view of the weather spreadsheet

that most people just use an empty row or column to separate different groups of cells. Therefore, $\sqrt{2}$ is chosen for the distance threshold k. That is, cells that are next to each other vertically, horizontally, and diagonally are in the same group. For a given *Cell(i,j)* (*i* is its column index, and *j* is its row index), there might be 8 adjacent cells: *Cell(i-1,j-1)*, *Cell(i,j-1)*, *Cell(i+1,j-1)*, *Cell(i-1,j)*, *Cell(i+1,j)*, *Cell(i+1,j-1)*, *Cell(i+1,j)*, and *Cell(i+1,j+1)* as shown in Figure 3.4.

Cell(i-1,j-1)	Cell(i,j-1)	Cell(i+1,j-1)
Cell(i-1,j)	Cell(i,j)	Cell(i+1,j)
Cell(i-1,j+1)	Cell(i,j+1)	Cell(i+1,j+1)

Figure 3.4: A cell's possible adjacent cells

Therefore, the goal of structural abstraction is to find a set of cell blocks. In each block, cells are all adjacent to each other directly and indirectly. Cells from different blocks are not adjacent to each other. The algorithm to structurally partition a worksheet is as follows.

Input: a list of cells from one worksheet

Output: a set of cell blocks

Method:

```
1. Create a hashtable based on the given list of cells (denoted as
     "cell hashtable");
2. Create a list to hold cell blocks (denoted as "block list");

3. WHILE the cell hashtable is not empty
4.      Create an empty cell list;
5.      FOR each cell in the cell hashtable
6.          Find the cell's adjacent cells in the cell hashtable;
7.          Insert the found adjacent cells into the cell list if
              they do not exist;
8.          Remove them from the cell hashtable;
9.          Recursively repeat step 6 for each currently found
              adjacent cell until no more adjacent cells can be found;
10.     END FOR
11.     Insert the cell list to the block list;
12.END WHILE
```

Computational abstraction

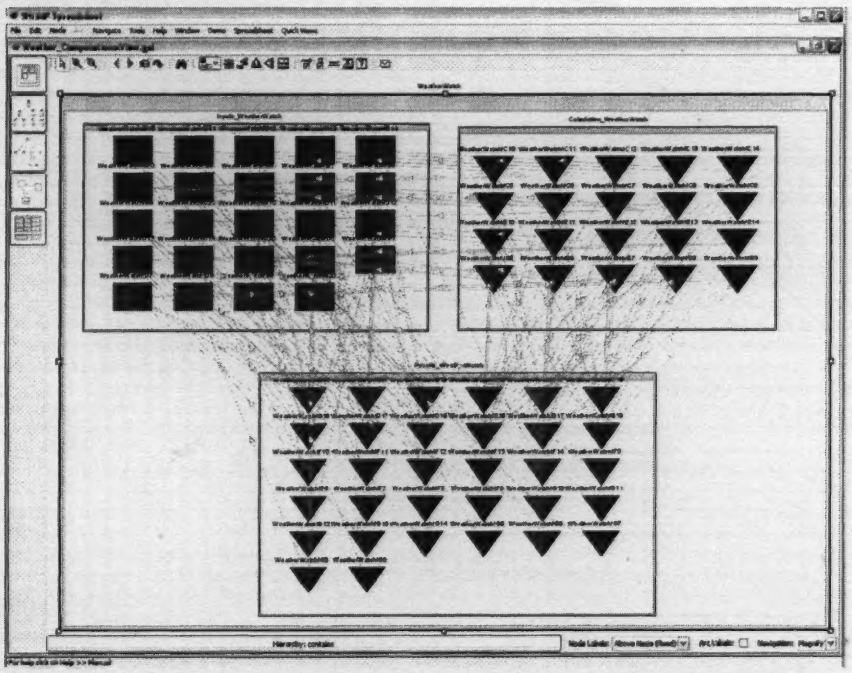


Figure 3.5: A structural view of the weather spreadsheet

Computational abstraction identifies high-level computational ranges such as input, calculation and output (result) blocks. In the computational abstraction, only input cells and formula cells are considered. Non-referred data cells such as label cells and stand-alone data cells are filtered out, because they do not contribute to any calculation. Figure 3.5

shows a computational view of the "weather" sample spreadsheet.

The goal of computational abstraction is to find all input cells and formula cells, and partition them into three blocks: inputs, calculation, and results. At the fact extraction stage, each non-empty cell has a corresponding GXL node. For each formula, each cell that it refers to has a corresponding GXL edge (pointing from the formula cell to the referred cell). Therefore, it is easy to filter out all the non-referred data cells; those cells that do not exist in any absolute or relative reference GXL edge must be a non-referred cell. The algorithm to partition a worksheet into computational blocks is as follows.

Input: a list of cells (GXL nodes) and a list of cell referencing pairs (GXL edges) from one worksheet

Output: input, calculation, and result cell blocks

Method:

```
1. Build a list of computational cells (input cells and formulas) by
     filtering out all cells that do not exist in any cell referencing
     pairs (GXL edges);
2. Create three cell blocks: inputs, calculation, and results;

3. FOR each cell in the created computational cell list
4.      IF the cell's GXL node type is not the formula type
5.          Insert it into the inputs block;
6.      ELSE IF the cell's reference appears in a cell referencing pair
          (GXL Edge), and the cell is a referred cell ("target" node)
7.          Insert it into the calculation block;
8.      ELSE Insert the cell into the results block;
9.      END IF
10.END FOR
```

Customized abstraction

Customized abstraction allows the user to build cell blocks based on their domain knowledge and preferences. First, the user needs to specify cell blocks in Excel (how Excel interacts with SHriMP-cell will be explained in Section 3.4.2). Then the Spreadsheet Analyzer will build blocks based on the user's inputs. Figure 3.6 shows a customized view of the "weather" spreadsheet.

3.3.6 Detecting anomalies

Anomalies are cells or ranges that might contain errors. Automatically detecting anomalies can significantly reduce the workload of spreadsheet auditing. Sajaniemi's absolute-direct and relative-offset (ADRO) cell referencing representation is adopted to detect anomalies

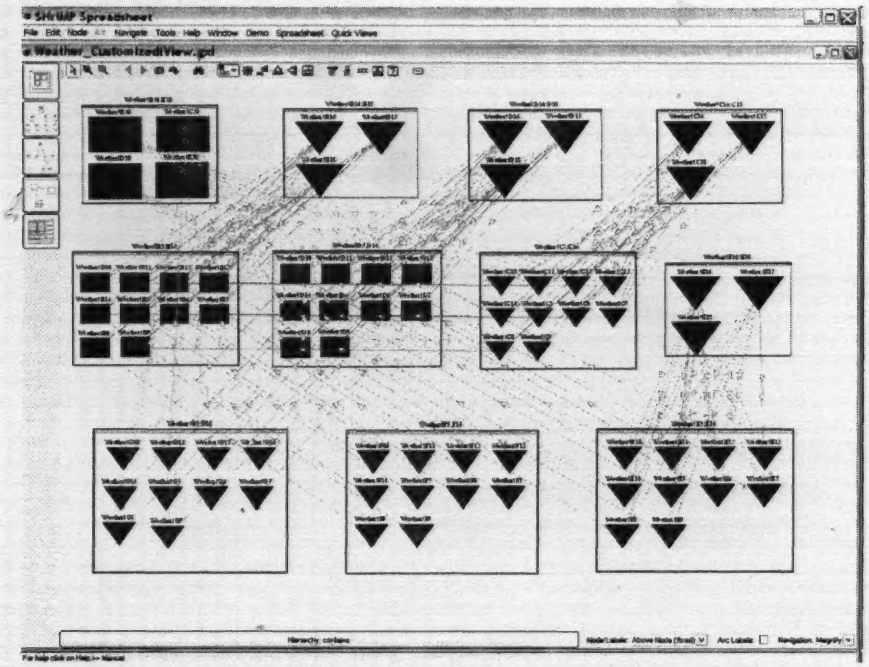


Figure 3.6: A customized view of the weather spreadsheet

among formulas which follow certain patterns [69]. The basic idea of ADRO is to represent a formula's absolute cell references using direct reference (the physical cell reference) and relative cell references using offsets to the formula's cell reference. In ADRO, a cell reference (for cells referred to by a formula) is a quadruple $\langle cs, c, rs, r \rangle$, where c and r are integers representing column index and row index, *cs* and *rs* are column and row symbols which may be either *A* (absolute) or *R* (relative). For example, for a given formula cell *E9* with a formula string *=D9-D$20*, the ADRO representation of *E9* will be: $= \langle R, 1, R, 0 \rangle - \langle R, 1, A, 20 \rangle$

ADRO representation isolates the formula's cell references from its physical location. Therefore, ADRO is very capable to compare whether or not a group of formulas follow the same "pattern". For example, in the "weather" spreadsheet, formulas in Range *E5:E14* follow the same pattern because their ADRO cell references are the same: $= \langle R, 1, R, 0 \rangle - \langle R, 1, A, 20 \rangle$. If *Cell E10*'s formula is changed to *=D11-D$20* by mistake, the ADRO cell references will be changed to: $= \langle R, 1, R, 1 \rangle - \langle R, 1, A, 20 \rangle$ which violates the pattern of its adjacent formulas. SHriMP-Cell can detect such anomalies and visualize them using an anomalous type of node as shown in Figure 3.7. Red is chosen to represent anomalous

formulas.

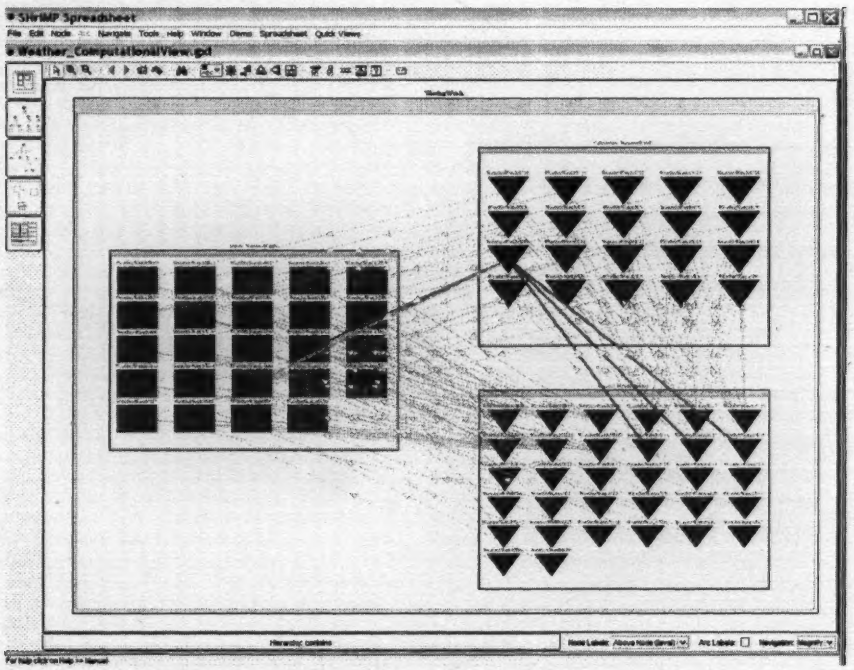


Figure 3.7: An anomaly detected in the weather spreadsheet

3.3.7 Generating GXL graphs

Finally, GXL graphs are built based on the extracted low-level facts and high-level abstraction information. SHriMP uses the "contains" type of GXL edge to represent nested graphs. Therefore, to put cell nodes, blocks, and worksheets in a hierarchy, "contains" edges need to be generated. For each worksheet, each non-empty block inside the worksheet will generate a "contains" edge pointing from the worksheet node to the block node. For each block, each cell node inside that block will generate a "contains" edge pointing from the block node to the cell node. Figure 3.8 shows part of the computational view GXL graph of the "weather" spreadsheet.

3.4 Tool integration

Tools should be integrated to improve usability and enhance the capabilities of each individual tools [51]. This section describes how SHriMP, the Spreadsheet Analyzer, and Excel are integrated together.

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE gxl SYSTEM "http://www.gupro.de/GXL/gxl-1.0.dtd">
<gxl>
    <graph id="Weather">
        <node id="WeatherWatch!B5">
            <type xlink:href="spreadsheet-schema.gxl#Data_WeatherWatch"/>
            <attr name="value">
                <string>-5.0</string>
            </attr>
        </node>
        <node id="WeatherWatch!C5">
            <type xlink:href="spreadsheet-schema.gxl#Formula_WeatherWatch"/>
            <attr name="value">
                <string>-23.0</string>
            </attr>
            <attr name="formula">
                <string>B5-B$20</string>
            </attr>
        </node>

        ... omitted ...

        <node id="Results_WeatherWatch">
            <type xlink:href="spreadsheet-schema.gxl#Block"/>
        </node>
        <node id="WeatherWatch">
            <type xlink:href="spreadsheet-schema.gxl#Worksheet"/>
        </node>
        <edge to="WeatherWatch!B5" from="WeatherWatch!C5">
            <type xlink:href="spreadsheet-schema.gxl#Relative-reference"/>
        </edge>
        <edge to="WeatherWatch!B20" from="WeatherWatch!C5">
            <type xlink:href="spreadsheet-schema.gxl#Absolute-reference"/>
        </edge>

        ... omitted ...

        <edge to="WeatherWatch!C7" from="Calculation_WeatherWatch">
            <type xlink:href="spreadsheet-schema.gxl#contains"/>
        </edge>

        ... omitted ...

        <edge to="Results_WeatherWatch" from="WeatherWatch">
            <type xlink:href="spreadsheet-schema.gxl#contains"/>
        </edge>
    </graph>
</gxl>
```

Figure 3.8: An example of partial GXL file

3.4.1 Integrating the Spreadsheet Analyzer with SHriMP and Excel

SHriMP is integrated with the Spreadsheet Analyzer by a menu extension as shown in Figure 3.9. Users can choose one of the three views to visualize a spreadsheet. Once a view option is chosen; a file chooser dialogue will appear that asks the user to pick an Excel file. Once an Excel file is selected, a GXL graph (named as the Excel file name followed by the view's name, and with a .gxl file extension) will be generated if the graph does not already exist and opened in SHriMP. If the graph already exists, another dialogue will appear asking the user whether or not to re-generate the graph.

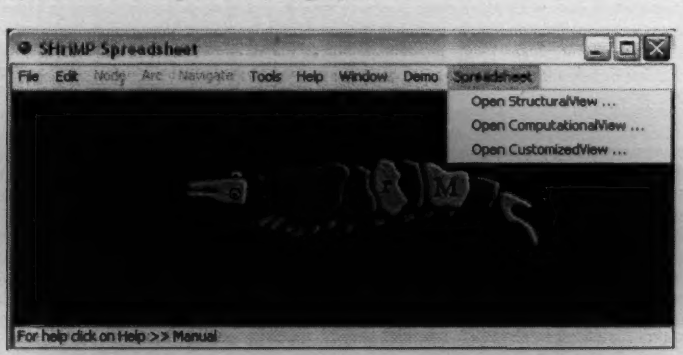


Figure 3.9: A spreadsheet menu extension to SHriMP

To extend the SHriMP menu, the class *ca.uvic.csr.shrimp.ShrimpApplication. StandAloneApplication.StandAloneApplication.java* needs to be extended. Each new menu item has a corresponding action which opens a file chooser. If an Excel file is selected, a *ShrimpApplication* object will be created based on the GXL graph generated by the Spreadsheet Analyzer.

3.4.2 Bridging SHriMP-Cell and Excel

As introduced in Section 3.2.3, SHriMP-Cell and Excel can "talk to each other" by reading and writing command files. The interaction features are accessible by adding an add-on menu and toolbar in Excel. This solution was proposed by Sajaniemi et al. in designing the S3 visualization technique [68]. Figure 3.10 shows the installer of the Excel extension.

To install the Excel extension, users just need to click on the "Install Excel Extension" button. Toolbar icons will be created as shown on the top left corner in Figure 3.10. The first icon is for performing requests from SHriMP; the second icon is for customized clustering of cells into blocks; the third icon is for removing existing customized clusters.

Do SHriMP request: For example, to select an Excel cell corresponding to a SHriMP

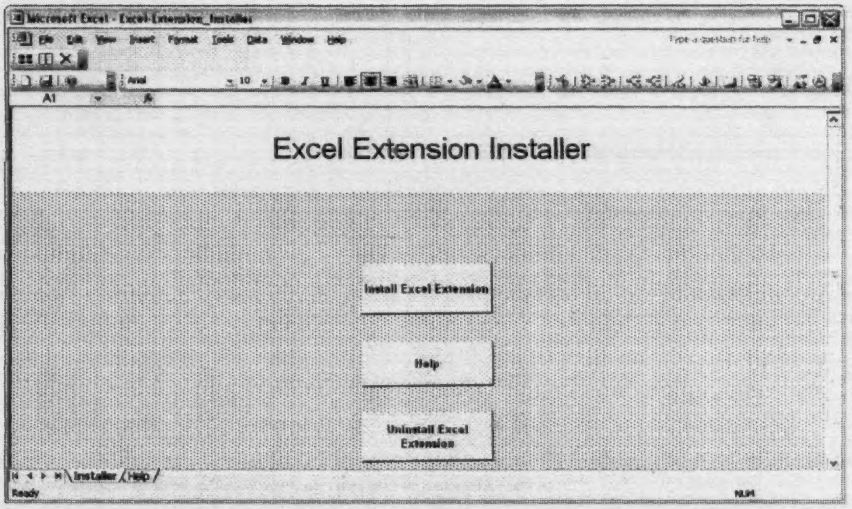


Figure 3.10: The installer program to install/uninstall the Excel extension

node, users first need to right-click on a data or formula node in SHriMP, and click on the "Show - Cells in Excel" menu item as shown in Figure 3.11. Then, users can switch to Excel and click on the "Do SHriMP Request" icon. The cell will be highlighted in Excel.

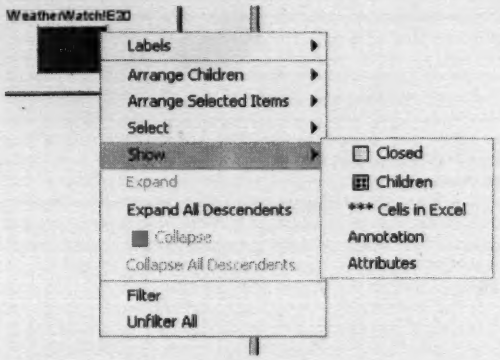


Figure 3.11: Pop up menu from SHriMP node

To extend the SHriMP pop-up menu, three classes need to be extended:
ca.uvic.csr.shrimp.PanelModeConstants.java,
ca.uvic.csr.shrimp.DataBean.AbstractArtifact.java, and
ca.uvic.csr.shrimp.DataBean.SimpleDataBean.SimpleArtifact.java

Add/remove blocks: As introduced in Section 3.3.5, the customized abstraction needs the user's input of cell clustering. This can be done by using the Excel extension as shown

in Figure 3.12. Users can group cells as a block, and specify the name of the block. The cell range reference is the default name of a block. Currently, blocks are not allowed to overlap.

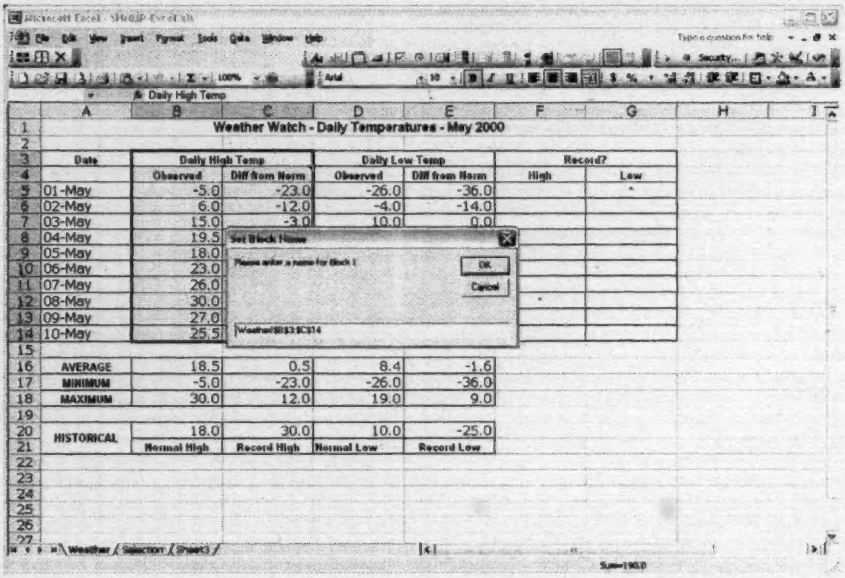


Figure 3.12: Add a cell block using the Excel extension

Block cell references are added to a ".blk" file so that the Spreadsheet Analyzer can read them to build customized views. If users want to build new blocks, they can remove all the existing blocks easily by clicking on the "Remove All Blocks" toolbar icon.

Chapter 4

Evaluation

Evaluation is important for software visualization tools because tools need to be evaluated to show their effectiveness. However, evaluation is often a challenging task, especially when time and resources are limited. Ideally, a tool should be evaluated for real-world projects with real users (field studies). This is usually hard to achieve in academia. Even in industry, the limited number of subjects and the variety of tasks restricts the usefulness of an evaluation. Therefore, various alternative evaluation techniques such as "quick and dirty" evaluations, controlled user studies, and predictive evaluations have been introduced [63]. Besides these "inexpensive" evaluation techniques, we also proposed an evaluation method based on perceptual theories, and it has been successfully used to evaluate the SHriMP visualization [76], and the layout of UML diagrams [84].

SHriMP-Cell is designed to assist end users to better understand spreadsheets. Thus, both the usability and effectiveness of the tool need to be evaluated. The goal of the evaluation is to validate whether this toolset is easy to learn, pleasant to use, and effective in helping to understand spreadsheet structures. This chapter introduces how SHriMP-Cell has been evaluated by guideline-based methods, a user study, and theory-based principles. Figure 4.1 shows how these different evaluation techniques are used in this thesis.

4.1 Guideline-based evaluation

As mentioned earlier, usability is one of the key issues that need to be addressed in end-user tool design and evaluation. However, many people, especially those from academia, find that many usability methods are often too expensive, too difficult, and time-consuming to use. Therefore, many "discount" usability inspection methods have been developed, such as heuristic evaluation, pluralistic usability walkthrough, claims analysis, cognitive walkthrough, feature inspections, etc. [58]. These usability inspection methods usually

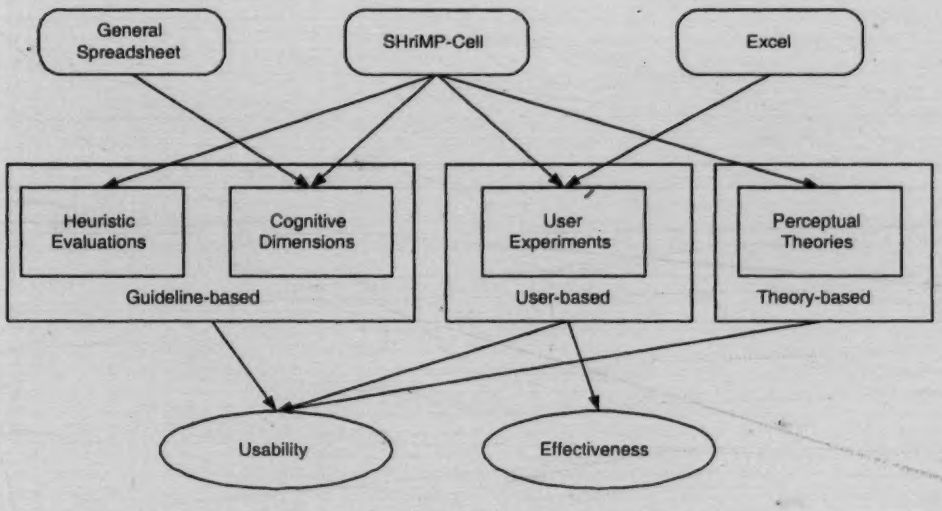


Figure 4.1: The evaluation framework in this thesis

only require a small number of users and are still effective [67].

4.1.1 Heuristic evaluation

Heuristic evaluation is adopted, because it is an inexpensive technique to evaluate the usability of a user interface. It only requires a few evaluators, yet it is still effective [59]. Nielsen found that the optimal number of evaluators for the best ratio of the benefits to the costs is three to five [58]. The ideal evaluators should be user interface design experts. Heuristic evaluation is used in this thesis work as a discussion technique, and as a question source for the user experiments.

The original criteria used in heuristic evaluation were presented in [52]. After a factor analysis of 249 usability problems, Nielsen revised the usability heuristics to the following list of criteria [57, 58]. We briefly explain each heuristic and add our toolset analysis.

1. ***Visibility of system status:*** *"The system should always keep users informed about what is going on, through appropriate feedback within reasonable time."*

   SHriMP-Cell has good visibility in general: the system usually keeps the user informed of what happened. This criterion will be further analyzed in Section 4.2.

2. ***Match between system and the real world:*** *The system should use the terminologies familiar to the user, rather than system-oriented terms. The system should follow well-accepted conventions, making information appear in a natural and logical order.*

The terminology used in SHriMP-Cell are common terms used in software and computer graphics.

3. ***User control and freedom:*** *The system should provide a clearly marked "emergency exit" to leave the unwanted state. The system should also support undo and redo.*

   SHriMP provides "Back", "Forward", and "Home" icons to ease navigation and to prevent users getting lost. "Back" and "Forward" bring the user to the previous and next state. "Home" can bring the user back to the start state.

4. ***Consistency and standards:*** *"Users should not have to wonder whether different words, situations, or actions mean the same thing."*

   Naming is consistent in SHriMP-Cell. For example, "cell" and "block" from Excel are also called "cell" and "block" on the SHriMP side.

5. ***Error prevention:*** *Good design should prevent a problem from occurring in the first place.*

   In SHriMP-Cell, the Excel extension prevents users from making a mistake by giving warnings. For example, if the user clicks on the "Remove All Blocks" icon, a confirmation dialog will appear asking the user to confirm the deletion.

6. ***Recognition rather than recall:*** *Things should be visible so that the user does not have to remember much information.*

   When a user mouses over any icon in SHriMP and the Excel extension, there is always a description about the function of that icon so that users do not have to remember.

7. ***Flexibility and efficiency of use:*** *Short cuts should be provided to speed up the interaction.*

   In SHriMP, many features have shortcut keys. For example, "h" will go to "home", "+" will zoom into a node, etc.

8. ***Aesthetic and minimalist design:*** *Dialogues should not contain any irrelevant or rarely needed information.*

   Dialogs in SHriMP-Cell are usually concise.

9. ***Help users recognize, diagnose, and recover from errors:*** *error messages should be defensive, precise, and constructive.*

Error messages in SHriMP-Cell are clear. For example, if the user tries to add a cluster that overlaps with existing clusters using the Excel extension, a dialog will appear with the error message, "clusters cannot overlap each other".

10. ***Help and documentation:*** *The system should contain instructive help and documentation.*

    In SHriMP-Cell, the Excel extension has a complete help document. While for SHriMP, since it has been evolving for many years, the help documentation is not up-to-date, and some features such as the "Query View tool" have no documentation. We added help documentation in the tutorial for all the SHriMP features used in the user study.

4.1.2 Cognitive dimensions

Heuristic evaluation is useful to evaluate the user interface of a system, and is usually performed by user interface design experts. However, spreadsheet systems might also have notational issues, and the evaluators in our user study are not experts in human computer interaction. To address the limitations of heuristic evaluation, cognitive dimensions are also adopted to evaluate spreadsheets and SHriMP-Cell. This is a framework (discussion tool) designed for people, including non-specialists, to evaluate the usability of information-based artifacts and notations [17]. Cognitive dimensions have been applied to evaluate the usability of spreadsheets and their extensions [18, 37, 42, 80].

In this section, we discuss how SHriMP-Cell addresses some spreadsheet usability issues from the cognitive dimensions perspective. This discussion is based on previous studies done using cognitive dimensions, our analysis, and observations from our user study which will be introduced in Section 4.2. A complete description of the cognitive dimensions framework is available in [33].

1. ***Viscosity*** *("resistance to change")*

   Viscosity should be minimized.

   Spreadsheet: not a problem for small changes because users can simply change cell content. However, larger changes that might affect the whole spreadsheet need more care.

   SHriMP-Cell: SHriMP-Cell can easily highlight cell dependencies both direct and indirect. Therefore, large changes in spreadsheets can be less problematic if cell dependencies have been well checked carefully before making the changes.

2. ***Hidden dependencies*** *("important links between entities are not visible")*

   Hidden dependencies should be minimized.

   Spreadsheet: there are hidden dependencies in spreadsheets, and it is hard to check all the cell dependencies at a glance. Checking indirect cell dependencies is even harder.

   SHriMP-Cell: SHriMP view shows all the direct cell dependencies at a glance, and the Query View can detect all direct and indirect cell dependencies for one or more nodes.

3. ***Visibility and juxtaposibility*** *("ability to view components easily")*

   Visibility and juxtaposibility should be maximized.

   Spreadsheet: visibility is satisfactory for spreadsheets in general, especially if users combine the spreadsheet normal view and the formula view.

   SHriMP-Cell: SHriMP-Cell can enhance visibility by showing all worksheets at the same time in the same graph.

4. ***Secondary notation*** *("extra information in means other than program syntax")*

   Secondary notation should be minimized.

   Spreadsheet: a trace feature can add arrows to a spreadsheet, which can be considered as a secondary notation.

   SHriMP-Cell: SHriMP view brings a separate secondary notation, which will be discussed further in Section 4.2.

5. ***Closeness of mapping*** *("closeness of representation to domain")*

   Closeness of mapping should be maximized.

   Spreadsheet: formulas and cell values are closely related to calculation needs, and they can describe the problem domain easily.

   SHriMP-Cell: the SHriMP view and the spreadsheet view can be closely mapped if the layout of SHriMP views can keep the spatial order of cells (e.g., layout by columns).

6. ***Progressive evaluation*** *("work-to-date can be checked at any time")*

   Progressive evaluation should be maximized.

   Spreadsheet: the progress in creating a spreadsheet can be easily checked at any time.

   SHriMP-Cell: SHriMP can be updated any time when the spreadsheets are changed, so the progress can be presented by SHriMP-Cell.

7. ***Hard mental operations*** *("high demand on cognitive resources")*

   Hard mental operations should be minimized.

   Spreadsheet: formula creation and tracing dependencies usually require hard mental operations.

   SHriMP-Cell: SHriMP-Cell can ease tracing cell dependencies and reveal computational data flow.

8. ***Diffuseness/Terseness*** *("verbosity of language")*

   Diffuseness should be minimized, and terseness should be maximized.

   Spreadsheet: notations in spreadsheets are concise.

   SHriMP-Cell: SHriMP's graphical view is also succinct because artifacts and relations are all embedded in one nested view.

9. ***Abstraction*** *("types and availability of abstraction mechanisms")*

   Abstraction should be maximized.

   Spreadsheet: Abstraction can be implemented using macros or VBA scripts.

   SHriMP-Cell: SHriMP-Cell provides three kinds of abstractions: structural, computational, and customized.

10. ***Role-expressiveness*** *("the purpose of a component is readily inferred")*

    Role-expressiveness should be maximized.

    Spreadsheet: role expressiveness in a spreadsheet depends on how well it is laid out, and how meaningful the labels are.

    SHriMP-Cell: the computational view improves role expressiveness by illustrating data flow.

11. ***Error-proneness*** *("notation invites mistakes")*

    Error-proneness should be minimized.

    Spreadsheet: it is easy to make mistakes in creating formulas and making cell references.

    SHriMP-Cell: The anomaly detection and trace dependency features in SHriMP-Cell can help reduce errors.

12. ***Perceptual mapping*** *("important meanings conveyed by position, size, color, etc.")*

    Perceptual mapping should be maximized.

    Spreadsheet: perceptual mapping in spreadsheets is normally weak because all cells look similar in terms of size, and shape, although good use of color might help.

    SHriMP-Cell: SHriMP-Cell conveys very good perceptual mapping by using color, size, and nesting, which will be discussed in Section 4.4.4.

4.2 User study

4.2.1 Goals

A user study was conducted to evaluate SHriMP-Cell based on real spreadsheet users. The main goal of the user study is to evaluate the usability of SHriMP-Cell, i.e., whether or not it is easy to learn, and pleasant to use. We also have the following goals besides the usability evaluation.

- Observe what strategies end users use to navigate and understand spreadsheets.
- Evaluate the effectiveness of SHriMP-Cell by comparing it with Excel 2003's formula evaluator tool (trace tool).
- Ask users questions from the heuristic evaluation and Cognitive Dimensions frameworks that are difficult to analyze.

The user study followed the design and procedure of Storey and Wong's study on evaluating software comprehension tools [75].

4.2.2 Pilot experiments

A pilot user, a professional software developer, was asked to use both Excel and SHriMP-Cell in two separate pilot experiments. The purpose of the pilot experiments was to find

potential problems in the user study early and improve its design. Through the pilot experiments, we realized that the training for each tool should be more detailed, and some tasks were too big to be finished under the time limit. Also, by reviewing the pilot study videos, inappropriate actions such as giving the user too many hints during the formal tasks were identified. We believe that the pilot experiments significantly improved the real user experiments.

4.2.3 Participants

Four participants (two males and two females) with a diversity of backgrounds were recruited at the University of Alberta. User A is a multimedia developer; User B is an e-learning design specialist; User C is a senior undergraduate engineering student minoring in business; User D is a chemistry graduate student. Each participant has at least 5 years experience with spreadsheets, and most of them have worked on over 200 spreadsheets. Prior to the experiments, each participant was asked to complete an on-line pre-study questionnaire to collect their experience with spreadsheets. Appendix B lists all the questions. The answers were compared to help divide the participants into two groups of similar experiments.

4.2.4 Experiment design

Two sets of experiments were designed to evaluate SHriMP-Cell and Excel. In the SHriMP-Cell experiments, users were allowed to use SHriMP-Cell and the formula view in Excel, but they were not allowed to use other auditing tools in Excel such as trace precedents and dependents. In the Excel experiments, users were allowed to use the formula view and the trace features, as well as common tools such as text search in Excel. The objective of running these two sets of experiments is to evaluate whether SHriMP-Cell is more effective than Excel in finding cell dependencies. Based on the users' expertise with spreadsheets collected through the pre-study questionnaires, two intermediate-advanced users were assigned to the SHriMP-Cell experiments. One advanced user and one beginner-intermediate user were assigned to the Excel experiments. Figure 4.2 shows how tool features are divided for the experiments.

In each experiment, there were six phases in total, and all of them were audio recorded. The training, practice, and formal task phases, which required users to use the computer, were also video recorded for further analysis.

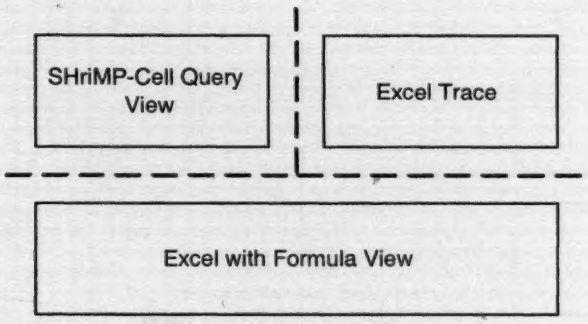


Figure 4.2: Two sets of experiments

4.2.5 Phases

Each user experiment took approximately 90 minutes and contained six phases: orientation (5 minutes), training (15 minutes), practice (15 minutes), formal tasks (40 minutes), post-study questionnaire (5 minutes), and post-study interview (10 minutes). For the SHriMP-Cell experiments, training and practice took a bit longer, because SHriMP-Cell has more new features for the user to learn than Excel.

Orientation: In the orientation phase, each user was informed of the purpose of the user study: to explore how end users understand spreadsheets, and evaluate tools to help end users in better understanding spreadsheets. Users' rights were reviewed and they were asked to sign the consent form in Appendix C. We stressed the point that this study is meant to evaluate tools, not users, to make sure the users felt relaxed and comfortable.

Training: In the training phase, basic spreadsheet concepts were reviewed, such as formulas, input cells, absolute and relative cell referencing, etc. A good understanding of these concepts is essential to finish the required tasks. A subset of tool features, which we believe necessary to finish the tasks, were demonstrated. The demonstration was based on a very small spreadsheet called "Macs.xls" that basically compares various prices of Macintosh computers. More detail of the training is described in the experimenter's handbooks in Appendix D and Appendix E.

Practice: In the practice phase, users were first asked to browse over a small spreadsheet called "Weather.xls", and try all the features of the tool introduced in the previous training session. This is to ensure that users have learned how to appropriately use the tool for comprehension tasks. Also, we observed how the users browsed over the spreadsheet, i.e., what strategies they use to understand the spreadsheet. Then, users were asked to do a few simple cell referencing and cell dependency tracing tasks to get them ready for the

formal tasks. During the practice session, users can ask any questions about the tool, or how to do a task if they have no clue.

Formal tasks: This is the main phase of the whole study, because observing how users use the tools to work on formal spreadsheet understanding tasks provides valuable feedback about the usability and effectiveness of each tool. Time is strictly limited to 40 minutes in this phase. The landline phone in the lab was disconnected and the users were asked to turn off their cell phones to minimize potential interruptions. Users can ask questions regarding tool features and clarification of the given tasks, but they were not supposed to ask questions on how to solve a problem. Tasks were given to each user one by one in the same order. The users did not see the next task until they finished or gave up the current one.

Post-study questionnaire: After the formal tasks phase, each user was given a list of questions regarding the ease of use, ability to find cell dependencies, and overall satisfactions of the tool. They were encouraged to finish the questions quickly based on their first impressions.

Post-study interview: In the last phase, users were asked informally about their experience using the given tool. They were encouraged to express any issues raised and provide suggestions to improve the tool.

4.2.6 Experiment variables

This subsection discusses some factors that might affect an experiment and the participants' performance.

Experiment environment: All the user experiments were conducted in the Software Engineering lab at the Department of Computing Science, University of Alberta. The computer used is an IBM PC workstation with Pentium 4, 2.4 GHz CPU and 1 GB RAM. The workstation has dual 18-inch monitors, and it is running Windows XP Professional. A video camera was set up to record the dual screens, over the participant's head. We observed that the dual-monitor is very useful for the study, especially for the SHriMP-Cell users, because they could see both the SHriMP view and the Excel view without switching back and forth. This observation is consistent with Storey's discovery that programmers nowadays often take advantage of larger screens and multiple monitors for complex tasks [72].

Chosen spreadsheets: Three spreadsheet documents were chosen for the user experiments. For the training, a very small spreadsheet was used which compared Mac computer prices, and it had two worksheets. For the practice, a small spreadsheet (18 rows and 7 columns in one worksheet) was used which lists the daily temperatures for a month

and computes the average, minimum, and maximum temperatures. This weather spreadsheet has only one worksheet ("WeatherWatch"), and it is taken from an education technology course [81]. For the formal tasks, a mid-size spreadsheet called "Historical-Income-Statement-and-Balance-Sheets" ("Income & Balance" for short, which has about 100 rows and 5 columns in two worksheets: "Finances" and "CashFlows") is taken from a financial spreadsheet modeling textbook [71].

4.2.7 Experiment results

It was very interesting to observe how users used the tools introduced in the training session to work on the formal tasks. The five formal tasks cover a wide range of spreadsheet comprehension work, including getting a high-level understanding of the given spreadsheet (Task 1), finding patterns in cell ranges (Task 3), tracing cell dependencies (Task 2 and 4), and identifying input cells (Task 5). Table 4.1 highlights how correctly each user finished each task. The scores are in a five-point scale from fail (1), bad (2), satisfactory (3), good (4), to excellent (5).

Table 4.2 shows how much time they spent on each task in minutes.

Table 4.1: Formal tasks correctness results

	SHriMP-Cell		Excel	
Tasks	User A	User B	User C	User D
T1	N/A	N/A	N/A	N/A
T2	5	5	4	4
T3	5	5	5	5
T4	5	3	4	5
T5	5	5	5	3
Total	20	18	18	17

Table 4.2: Formal tasks time spent

	SHriMP-Cell		Excel	
Tasks	User A	User B	User C	User D
T1	5.5	7.0	7.0	12.0
T2	6.0	13.0	3.5	4.0
T3	3.0	4.5	3.5	3.0
T4	5.0	7.0	13.0	11.5
T5	6.0	4.0	12.0	9.0
Total	25.5	35.5	39.0	39.5

Formal Task 1

Browse the given spreadsheet and explain what it does, and how it is structured. In addition, explain how the calculation works?

This task allows the experimenter to observe what strategies end users use to understand spreadsheets. To get a high-level understanding of the given spreadsheet, most users started from reading the headers and labels to get a sense of what a spreadsheet is about (the top-down strategy), and quickly switched between the normal spreadsheet view and the formula view to see where the formulas were and how many of them there were. However, one user started from reading formulas first (the bottom-up strategy). He said that headings and labels could be vague and misleading sometimes, while formulas can always tell you how things are done. Domain knowledge, i.e., knowledge in finances in this study definitely helped to understand the given spreadsheet. A couple of users mentioned that it would be much easier for them to understand the "Income & Balance" spreadsheet if they had some finance and business background. This observation is quite consistent with the results from a literature review that end users use similar strategies to understand spreadsheets as professional programmers do in understanding software systems, as described in Section 2.4.

One interesting observation is that most users only looked at a portion of the spreadsheet and guessed the rest. One of the users said he usually looked at things in blocks, trying to understand one block, and guessing other blocks. The users made educated guesses, because most documents follow certain patterns and conventions. Although guessing helped, sometimes it could be misleading. A couple of users realized that their initial guesses were wrong, and they had to look at labels and formulas carefully to "re-understand" the spreadsheet. Guessing works well when the spatial layout of the spreadsheet is intuitive and consistent.

Most users spent about six to seven minutes on this task, and only User D spent twelve minutes. She read all the labels carefully, and tried to understand the whole spreadsheet in detail. Her deep understanding helped her to later finish Task 3 easily. SHriMP-Cell users did not spend much time browsing the SHriMP graphical view of the spreadsheet. One of them said this is because SHriMP-Cell only showed cell references instead of labels and values at a glance, and cell references cannot help much in understanding what the spreadsheet does. However, SHriMP-Cell users could quickly identify how data flows in the spreadsheet, and they could easily tell that the CashFlows worksheet is a computational

sheet, i.e. has no input cells. There was no score for this task, because it was meant for us to observe how users understand the given spreadsheet.

Formal Task 2

Suppose you changed the value of cell Finances!B27, what other cells' value might also have been changed?

This task requires the user to find the dependents of cell Finances!B27, i.e., cells that refer to or depend on Finances!B27. There are only 4 cells in total depending on it directly and indirectly: Finances!B31 (direct), Finances!B40 (indirect), CashFlows!C33 (direct), and CashFlows!C35 (indirect).

In SHriMP-Cell, this task can be done by using the Query View tool. There are two ways to query a node's neighboring nodes. A user can select a node first, and a query can be done based on the selected node, or a user can search for a node based on its cell reference, and query on the search result. Figure 4.3 shows the results obtained from the Query View in SHriMP-Cell.

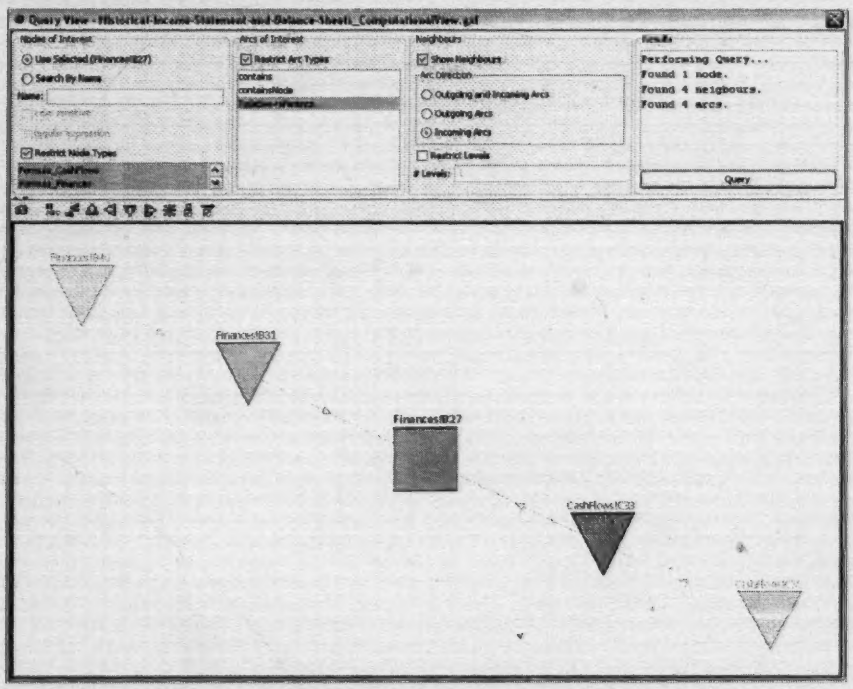


Figure 4.3: Results from the Query View tool for Task 2

Both SHriMP-Cell users quickly realized that they should use the Query View Tool to

answer this question. One user used the first way: he used the SHriMP built-in search tool to find the cell, selected it, and then did the query. He struggled a bit with whether to choose the "Incoming Arcs" or "Outgoing Arcs" option. He found the right answer at the end, but was not very confident because he was new to the tool. He recommended that it would be nice if the Query View nodes can be zoomed in to see a formula in the same way as the main SHriMP view. To validate the result, he had to switch back to the main SHriMP view, and zoomed into one of the result nodes. The formula in the node showed that his result was correct. The other SHriMP-Cell user used the second way, but did not get any results due to wrong query options. She did not refer to the mini tutorial provided in the training phase, but just tried changing query options. She eventually found the answer, and this task took her 13 minutes.

In Excel, both users used the trace tool to trace dependents of Finances!B27. However, if a dependent appears in another worksheet, Excel can only show an external sheet icon as shown in Figure 4.4. The users had to go to the other worksheet, and search for the cell reference manually. Both Excel users missed an indirect cell dependent: one user missed tracing the indirect dependents in the Finances worksheet, and the other user missed tracing the indirect dependents in the CashFlows worksheet.

SHriMP-Cell and Excel use different arrow directions to show cell dependencies as shown in Figure 4.5. Excel's arrow direction shows how data flows, and SHriMP-Cell's arrows show the direction of cell dependencies.

Formal Task 3

In the "CashFlows" worksheet, the column of year 1999 is blank. Do you think this is a mistake or something is missing or unfinished here? Why or why not?

This task requires the users to check the patterns of the columns next to the year 1999 column. The answer is that it is not a mistake or something is missing. Column 1999 needs to refer to the previous year's data, which does not exist. Although the first two cells could be calculated, they do not make too much sense if they are listed without the rest of the column.

No user actually used SHriMP view for this task, because SHriMP-Cell does not show formulas and values at a glance. All four users gave the right answer, and the average time spent on this task was 3.5 minutes. Two users initially thought that it was a mistake or something was missing in the year 1999 column, but they changed their minds after checking the formulas more carefully. The user who spent much time on Task 1 already

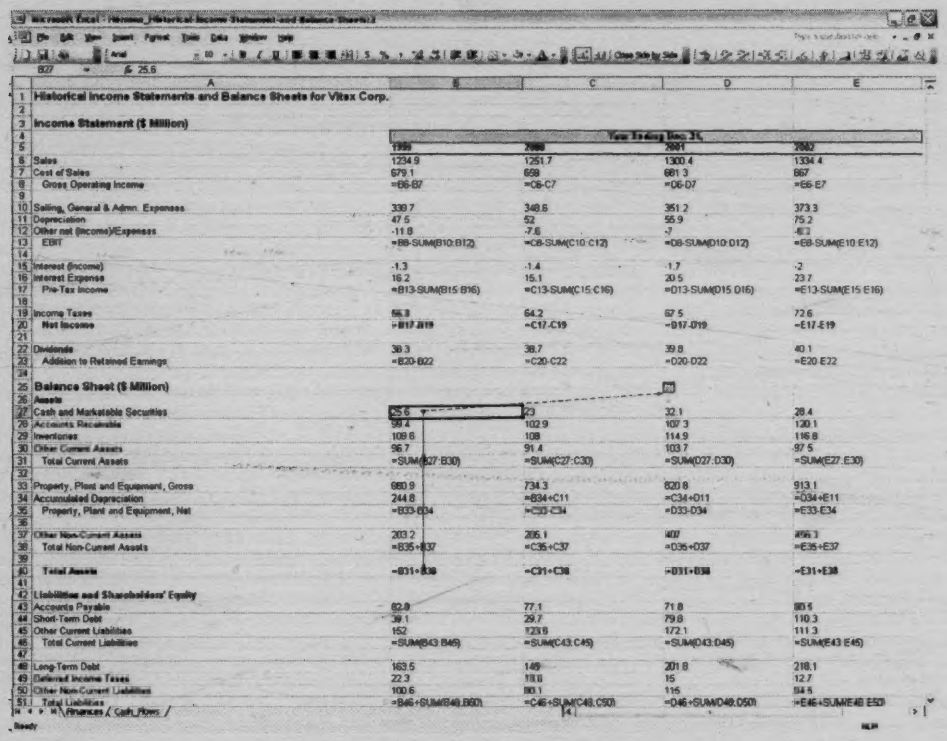


Figure 4.4: Results from the trace dependents tool in Task 2

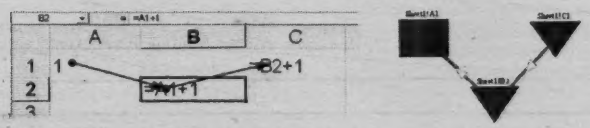


Figure 4.5: Arrow directions in Excel and SHriMP-Cell

wondered about why year 1999 was blank when she did Task 1. Since she was very familiar with the whole spreadsheet, she quickly found the answer for this task.

Formal Task 4

What cells does cell "CashFlows!E16" depend on (i.e., refer to) directly and indirectly?

This task is similar to Task 2, but the query or trace is in the opposite direction. The result contains 35 nodes which is time-consuming for the Excel users to trace because they have to click on the trace icon multiple times, and switch between two worksheets to find indirect precedents.

In SHriMP-Cell, the users still struggled with the arc direction option. Both users re-

quired more explanations of the arc directions. User A found the right answer, but User B chose the wrong arc direction and only found one cell node. The average time to finish this task for the SHriMP-Cell users was six minutes. The correct result from the SHriMP Query View tool is shown in Figure 4.6. Users can easily tell the level of indirect reference for each cell in the hierarchy.

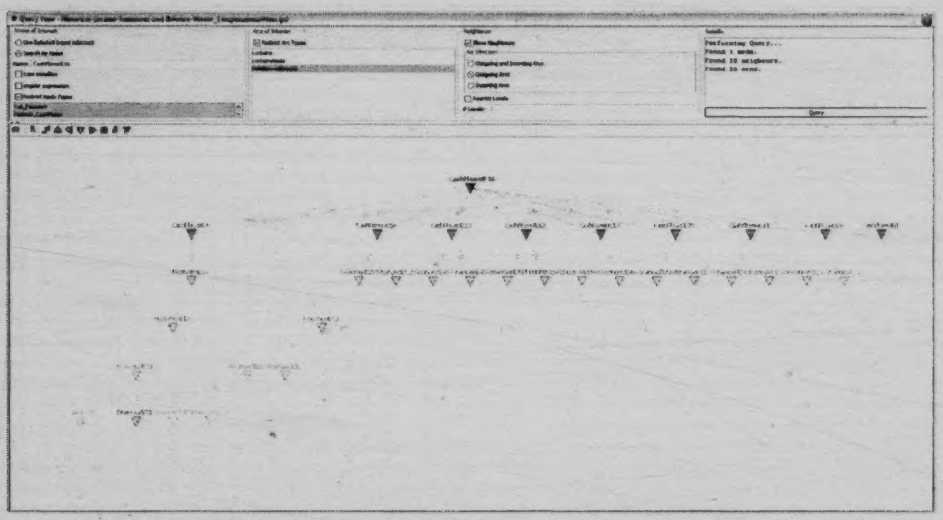


Figure 4.6: Results from the Query View tool for Task 4

In Excel, both users started by looking at the formula views, and quickly found the direct precedents. However, finding the indirect precedents was not easy, because all of them are on the Finances worksheet as shown in Figure 4.7. User C used the trace tool by multiple clicks to trace indirect precedents. He found all the precedents, but he also listed a wrong cell range in his result. User D did not use the trace tool. She said that too many arrows confused her. She just read the formulas one by one, and found the precedents manually. She was very careful in checking cell references, and checked her answers twice. Her answer was complete as shown in Figure 4.12. The average time spent by the Excel users was about twelve minutes.

Formal Task 5

Find all the initial input cells and list their cell and/or range references.

Input cells are data cells that are referred by other formulas, i.e., non-formula cells that contribute to the computation.

In SHriMP-Cell's computational view, it is very easy to find all the input cells, because

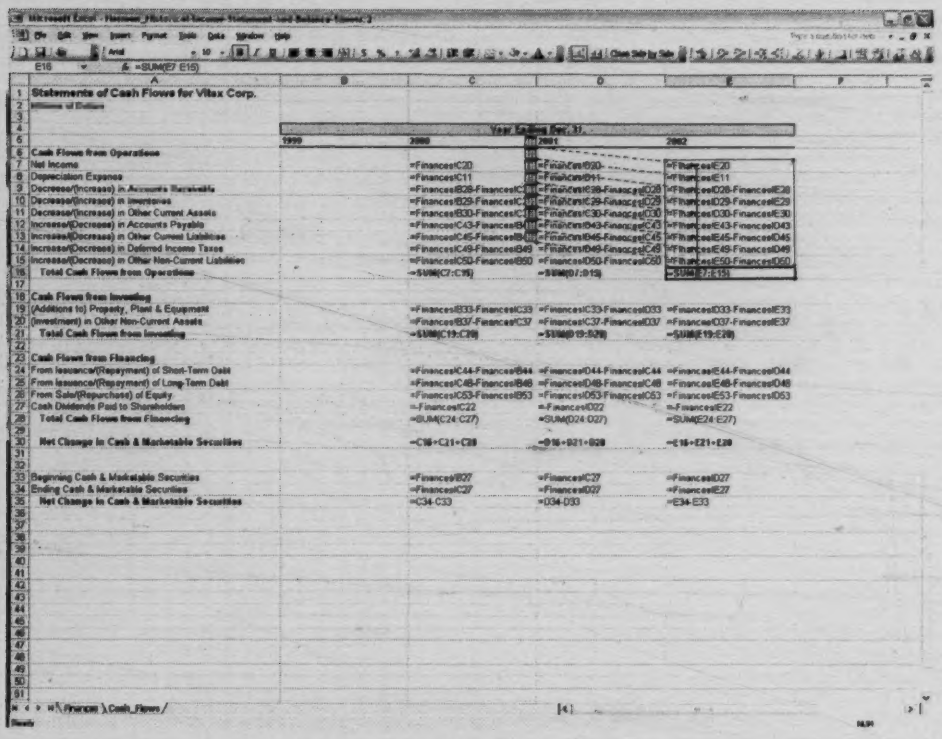


Figure 4.7: Results from the trace precedent tool in Task 4

non-input data cells are filtered out, and all input cells are in the same block. Both SHriMP-Cell users successfully found the input cells, and the average time for them to finish this task was five minutes.

In Excel, finding all the input cells is more time-consuming, because the users have to make sure that the data cells are referred by other formulas. The two Excel users used two different strategies to find the input cells. User C found the data cells first, and used tracing to check whether they were referred by other formulas. Excel does not support tracing a group of cells. He had to trace the dependents of data cells one by one. He found all the input cells after checking the results twice. User D used the trace precedents feature to check formulas. However, she listed many formula cells that she thought were input cells, and she did not have enough time to finish this task. The average time for the Excel users spent on this task was eleven minutes.

Post-study questionnaires

The post-study questionnaire questions basically focus on the users' experience and satisfaction using the tool. All questions are in a five-point scale from strongly disagree (*), disagree (**), neutral (***), agree (****), and strongly agree (*****). A complete list of the questions is available in Appendix F. Table 4.3 shows the results:

Table 4.3: Post-study questionnaire results

	SHriMP-Cell		Excel	
Questions	User A	User B	User C	User D
Q1	**	****	****	****
Q2	****	*****	*****	*****
Q3	****	****	****	*****
Q4	****	****	*****	****
Q5	*	****	****	*****
Q6	***	***	*****	*****
Q7	**	****	*****	*****
Q8	***	****	****	*****
Q9	**	*****	*****	*****
Q10	***	****	****	****

The results show that User B, C, and D are quite satisfied with the tools, while User A is not very satisfied. In particular, User A strongly disagreed with Question 5: "The interface of this tool was pleasant to use". Ironically, User A was the only user who perfectly finished all the tasks within the shortest time (25.5 minutes).

Interviews

The interviews were valuable to explain the results of the post-study questionnaire. They also allowed us to collect answers about the questions we had in Section 4.1. A complete list of the interview questions is attached in Appendix G.

For SHriMP-Cell, both User A and B agreed that SHriMP-Cell is easy to learn and use in general, and it is very powerful to find relationships between cells. User A said that this tool would be very handy to find relationships in complex spreadsheets. User B expressed her opinion of SHriMP-Cell: "the tool has more (features) than I expected, and it is less complicated than I thought". Both users pointed out that the terminology used in SHriMP-Cell is somehow confusing to them. For example, User A did not like the term "arc", as arc to him should be curvy, while all the "arcs" in SHriMP-Cell are normally straight by default. User B also mentioned that it would be hard to use the tool without dual monitors. User A explained why he was not satisfied with the tool. He did not like the slow response, and he

was not confident with his results because he was new to the tool. Since he is a professional graphic designer, he was very picky about the interface. He tended to compare this tool with professional graphic design tools such as Adobe Photoshop [13] when he evaluated SHriMP-Cell's usability. For example, when he was in the zooming mode in SHriMP-Cell, he expected that the tool should provide a little "grabber" mouse pointer so he could drag the viewed portion easily.

For Excel, both users claimed that they were satisfied with the tool. It is easy to learn and simple to use. The formula view is especially helpful for them to check range patterns. Both users experienced the inconvenience of not being able to trace precedents or dependents that are on another worksheet. To find these, they had to switch to the other sheet and find the cell references manually. User C suggested that it would be nice if the Excel trace toolbar icons have short-cut keys. Often, users have to move the mouse pointer a lot to trace multiple cells and multiple levels of dependencies. Also, he would like to be able to trace precedents or dependents for a range of cells, rather than just one single cell at a time. However, we suspect that the lack of support to trace multiple cells in Microsoft Excel must have its reason, perhaps because the results of the trace for multiple cells would be too "messy" visually.

4.3 Discussion

Both SHriMP-Cell users struggled with the arc direction option in the Query View tool, even though it was introduced and listed in the mini tutorial. This might be because the arc directions used in SHriMP-Cell are opposite to arrow directions in Excel. Arc direction in SHriMP-Cell means the direction of dependency, while arrow direction in Excel means the direction of how data flow.

Dual screens are essential in effectively using SHriMP-Cell. Both SHriMP-Cell and Excel users took advantage of the dual monitors. SHriMP-Cell users can put the SHriMP view on one screen and the Excel window on another screen to check them at the same time. Excel users can lay two worksheets side by side using two screens so that they can easily switch between two worksheets to trace dependencies across worksheets.

Limitations: Here are limitations of the conducted user study.

- The sizes of the spreadsheets used in the experiments are relatively small compared to real-world spreadsheets. Still, the spreadsheet chosen for the formal tasks is not simple; there are many cell dependencies crossing worksheets. If the spreadsheets

were too big, users would not be able to learn the tool features easily, and finish the tasks under the time limit.

- The number of participants is relatively small, not enough to even out different background and domain knowledge. Thus, the results cannot be used for statistical analysis. However, we believe the users have many years of experience with spreadsheets. Therefore, they should be able to validate the usability and effectiveness of SHriMP-Cell.

- We tried to use some criteria from heuristic evaluation in our post-study questionnaire and interview questions. However, most of the participants are not user interface design experts. Therefore, their answers might not be very solid.

4.4 Theory-based evaluation

To further analyze the usability of SHriMP-Cell, perceptual theories can be applied to evaluate the readability of the SHriMP views, because perceptual factors are important for effective information visualization [35]. In this section, we outline several fields in cognitive science, such as the theory of perception, perceptual organization, and perceptual segregation. The laws of perception explain how our visual system identifies objects and how we put together basic features to observe a coherent, organized world of things and surfaces. From a software visualization perspective, the principles of perceptual organization provide the basic design rules to organize multiple artifacts, so that users can group related information and segregate useful information easily and without ambiguity. Perceptual theory can also be applied to evaluate the effectiveness of software visualization tools [76, 82]. In our previous work, we have successfully evaluated the SHriMP visualization [76] and the layout of UML diagrams [77, 84]. To evaluate SHriMP-Cell, perceptual principles can be applied to validate whether the visualization is readable and potentially understandable.

4.4.1 Theories of perception

It is widely accepted that visualization helps people to understand information, but how the brain processes, transforms, and interprets visual stimuli is still unclear. Various theories of perception have been proposed [62].

- **Marr's theory** [47]:
  Cognitive functions are filters that operate on raw visual stimuli and turn them into information.

- **Gibson's theory** [31]:

  We shift our attention based on the structure of the environment and create a cognitive map to interact with the world.

- **Gestalt theory** [16, 53]:

  We restructure our perception in ways that make it unified and coherent. This theory formulates the principles of organization to explain why some displays are better than others. Perceptual organization will be discussed in more detail in the next subsection.

- **Theory of notation** [62]:

  Many researchers try to find good notations for visualizations, including symbol systems in graphs to help convey semantics.

4.4.2 Perceptual organization

Perceptual organization refers to how objects that we perceive in the world are located and related with one another [22]. Research in perceptual organization studies how small elements are grouped into larger objects [32], which is important for tool designers to devise a clear visualization of a software system. The following are several important principles of perceptual organization, most of which are from the Gestalt Laws [22, 32].

- **Law of good figure (Prägnanz)**:

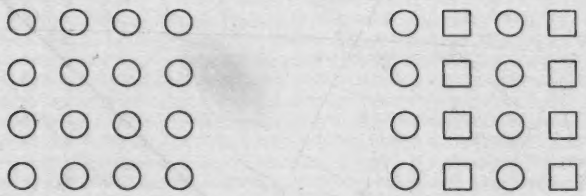
  Prägnanz is a German word which means "suggestive figure". The law of *good figure* is also called the law of *simplicity*. That is, images are perceived in a way such that their structures are as simple as possible. The "goodness" is sensitive to the amount of information necessary to describe a figure. Usually, a simpler and more stable figure is considered a "good" figure [70].

- **Law of similarity**:

  Similar elements (e.g., common shape or color) appear to be grouped together. For example, in Figure 4.8, (a) can be perceived as either horizontal rows or vertical columns of circles, but (b) would most likely be perceived as vertical columns of squares and circles because of the similar shapes.

- **Law of continuation**:

  Points tend to belong together if they result in straight or smoothly curved lines when



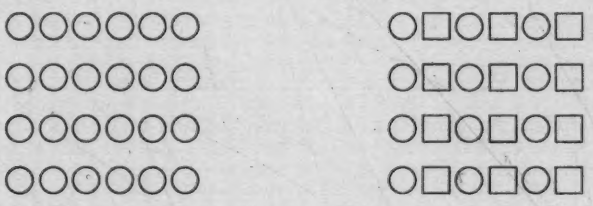
(a) (b)

Figure 4.8: Examples of the law of similarity.

connected, and lines are grouped together in such a way as to follow the smoothest path.

- **Law of proximity**:

  Elements that are close to each other are grouped together. For example, in Figure 4.9 (a), the circles are more likely to be perceived as horizontal rows since they are closer horizontally. In Figure 4.9 (b), the shapes are still more likely to be perceived as horizontal rows because the law of proximity overpowers the law of similarity discussed before.



(a) (b)

Figure 4.9: Examples of the law of proximity.

- **Law of connectedness**:

  Elements that are physically connected are perceived as a unit. In Figure 4.10, we perceive three dumbbells rather than some pairs of dots. Note that the dots that are next to each other in adjacent dumbbells are actually closer together, and according to the law of proximity, they should be grouped together. However, in this case, the law of connectedness overpowers the law of proximity.

- **Law of common fate**:

Figure 4.10: An example of the law of connectedness.

Things that are moving in the same direction will be grouped together.

- **Law of familiarity**:

  Elements are more likely to be grouped together if the groups seem familiar or meaningful.

4.4.3 Perceptual segregation

Compared to perceptual organization, research in perceptual segregation basically studies the problem of "figure-ground segregation" to determine when objects (figures) are seen as separate from the background (the ground) [32]. The fundamental figure-ground theory was proposed by Gestalt psychologists, who believed that a figure is more familiar and memorable than the ground. They also stated that a figure is perceived as being in front of the ground, and the contour of a figure and the ground more likely belongs to the figure. Experiments show that the following factors make an object more like a figure.

- **Law of symmetry**:

  Symmetric areas are usually seen as a distinct figure.

- **Law of orientation**:

  Horizontal or vertical orientations have higher probabilities to be seen as a figure than other orientations. For example, in Figure 4.11, the vertical/horizontal "plus" propeller shape is more likely to be perceived as figure than the tilted "cross" shape.

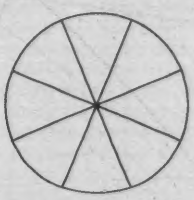


Figure 4.11: An example of the law of orientation.

- **Law of contour**:

Modern theories about figure-ground segregation discovered that contours (i.e., boundary edges) help in figure-ground perception [44].

4.4.4 Applying perceptual theories

In this section, we discuss how perceptual principles can be applied to evaluate the SHriMP tool in the SHriMP-Cell toolset.

Law of good figure (Prägnanz): To produce a "good" figure, SHriMP has a powerful show/hide feature: certain types of nodes and arcs can be shown and hidden. A node can be collapsed to hide all its child nodes and related arcs.

Law of similarity: SHriMP-Cell takes full-advantage of shape and color to separate different kinds of nodes and arcs. Different shapes are used for data cell nodes, formula nodes, and comment nodes. Different colors are used for nodes belonging to different worksheets. The users can easily tell at a glance what type a node is, and which worksheet it is from.

Law of continuation: Arc crossing is a problem in SHriMP, especially when there are many arcs. Too many crossings make the arcs less continuous and hard to follow, which violates the law of continuation.

Law of proximity: The objective of the abstractions is to put "closely related" nodes together. For example, in the structural view, all the cell nodes that are spatially close together will be grouped into one block.

Law of connectedness: Nodes do not overlap in SHriMP. However, node labels overlap as the number of nodes increases. This makes the label hard to read. SHriMP's solution is to allow users to choose whether or not to show the node labels, as well as adjust the font and size of the labels.

Law of common fate: Some animations in SHriMP follow the common fate principle. For example, when the user changes the layout to the radial layout, the artifacts that will be grouped together move in the same direction. This explains why User A was so fascinated with the different layout options.

Law of familiarity: Spreadsheets use grids to organize data which is familiar to people. SHriMP views might not be familiar to most spreadsheet users.

Law of orientation: SHriMP's node layout is based on a grid. Nodes are ordered in rows and columns by default which make them easy to group.

Gestalt theory: Gestalt theory states that the whole is more than the sum of its parts [22, 70]. SHriMP-Cell's computational view shows the dataflow as a whole, which is easier to

understand than the cell level dependencies if the user wants to get a high-level sense of how a calculation works.

Task 4

What cells does Formula Cell "CashFlows!E16" depend on (i.e., refer to) directly and indirectly?
Note, to save time, if the cell is in the "CashFlow" sheet, you don't need to write the sheet name. If the cell is in the "Finances" sheet, you can just underscore the cell reference.

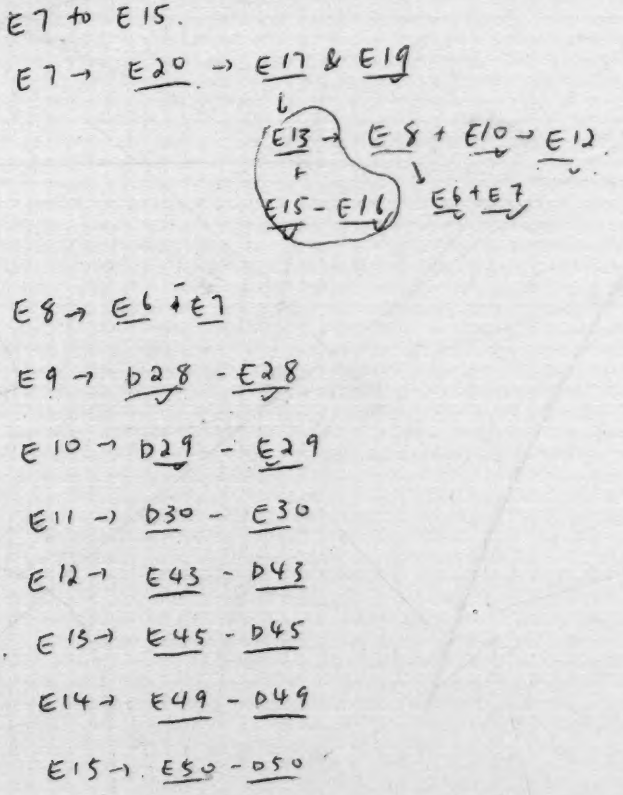


Figure 4.12: User D's answer to Task 4

Chapter 5

Related work

5.1 Spreadsheet visualization

Researchers have explored different ways to visualize spreadsheet documents to improve understandability:

CogMap [37]: To improve the communicability of spreadsheet documents, Hendry and Green designed a "cognitive map" representing a spreadsheet's structure and tried to use it to fill the gap between the tabular view and the computational layer. The authors believe that CogMap is closer to a user's cognitive representation of spreadsheet documents. Therefore, such a simple tool can improve the comprehensibility and communicability of existing spreadsheets. CogMap allows the user to annotate spreadsheet documents with tags ("types") and links ("relationships") in various colors. Users can arbitrarily define the display color, specify the line style, and show/filter tags and links.

S2/S3 Visualization [69]: Sajaniemi introduced the "formulae, formats, relations (FFR) model" to formalize spreadsheet documents and designed the S2 and S3 visualizations for spreadsheet auditing. The S2 visualization identifies areas that contain equal type values, equal format, and similar formulas. References between areas are also identified, and arrows are drawn to represent range references. The S3 visualization improves the S2 visualization by using dashed lines to separate subareas, and arrows for subareas are unbroken to reduce the number of arrows.

Fluid Visualization [41]: Igarashi et al. introduced a few techniques to visualize spreadsheet documents to ease the accessibility of formulas and dataflow structures during maintenance. The fluid visualization provides both local views to visualize dataflow structures associated with individual cells, and global views to present the dataflow for the entire worksheet.

Ballinger's Visualization [14]: Ballinger et al. used various visualization techniques

such as utilization diagrams, clustering views, data dependency flow, fisheye view, and formula view. In the "real-estate utilization diagram", any coordinate where cells appear is assigned a colored circle similar to a heat-map effect. The data dependency flow diagram shows the trend of how the data flows. Clustering visualizations is based on spatial relationships between cells. The precedent cell dependency diagram represents Excel's trace precedents feature in a different way.

5.2 Spreadsheet auditing

Research in spreadsheet auditing focuses on finding irregularities or anomalies, i.e., areas that might contain errors in spreadsheets. These dangerous spots need to be carefully checked. The auditing technique shrinks the testing scope of spreadsheets so that errors can be revealed effectively.

Clermont's Visualization [24, 25, 26, 27]: Clermont proposed a spreadsheet auditing technique based on the logical structure of a spreadsheet document, and clusters cells based on equivalence classes such as copy equivalence, logical equivalence, and structural equivalence. He also designed a structure browser to visualize different classes of cells. His auditing approach and toolkit have been evaluated in various contexts.

Header Inference [12, 29]: Erwig and his colleagues proposed a unit and header inference approach to cluster spreadsheet cells. They implemented header assignment views in Excel for spreadsheet auditing.

5.3 Tool comparisons

This section compares SHriMP-Cell with other spreadsheet visualization tools, including CogMap, S2/S3 Visualization, Fluid Visualization, and Ballinger's Visualization as introduced in Section 5.2.

Distinct features of SHriMP-Cell:

Here are the distinct features of SHriMP-Cell:

- SHriMP-Cell uses a reverse engineering technique, and the views are based on fact extraction from the current version of a spreadsheet document. Therefore, the views are always up-to-date when a document changes.

- Three different views are proposed and implemented to fit different comprehension needs.

- The Spreadsheet Analyzer has been integrated with SHriMP and Excel. Users can use SHriMP, the Spreadsheet Analyzer, and Excel together as a whole.

- SHriMP-Cell takes full advantage of SHriMP for visualization, which supports various comprehension strategies. SHriMP has powerful filtering functions such as the show/hide feature, dependency analysis such as the Query View, and zooming to show different levels of abstraction.

Comparison to CogMap: CogMap allows users to annotate spreadsheets by introducing types and relations. However, it cannot automatically extract any information such as data cells, formulas, comments from existing spreadsheet documents as SHriMP-Cell does. All the operations have to be done manually by the end-user, which might be tedious and time-consuming for large documents. For example, if users want to annotate all the cells referred by a formula, they have to look at the formula and find all the cells and/or blocks involved in the formula one by one.

Comparison to S2/S3 Visualization: S2/S3 visualizations can be seen as extensions to Excel's trace feature. They draw arrows between cells and ranges to highlight computational dependencies. However, S2 and S3 are based on Excel's tabular view, which is hard to express high-level abstractions. Also, when there are many arrows on the tabular view, it is hard to read because of the overlapping. Furthermore, S2 and S3 visualizations still cannot visualize data dependencies across worksheets.

Comparison to Ballinger's Visualization: Ballinger's visualization also uses reverse engineering to identify cell dependencies. Ballinger adopts a few different visualization techniques as shown in Section 5.1. However, these visualizations seem hard to map to the spreadsheet view, and no evaluation has been performed to validate the usefulness of these visualizations. Furthermore, no interactions are available between the visualizations and the spreadsheet application.

Chapter 6

Conclusions and future work

6.1 Lessons learned

During the design, implementation, and evaluation process, we had successes and failures. The following lessons were learned during this thesis work:

- Third-party libraries and APIs need to be well tested before adoption. In our early design stage, we chose POI to read Excel files, because it was very popular in the Excel Java developer community. However, after testing it on more complex spreadsheets, we found that POI could not fit all our needs as we expected, because the way we use POI is quite different from most Java developers. Therefore, JXL, a less well-known API, was also adopted and extended to address the limitations of POI.

- "Thinking aloud" should be balanced in user studies. Thinking aloud is the technique we adopted to observe how users understand spreadsheets. However, if we interrupt the users too often to encourage them to talk, it would be hard for them to focus on performing complex tasks, increasing the time they spend on each task and creating a potential bias.

- End users are often over-confident about their work. Error prevention is important in designing end-user tools. We observed that end users often neglect testing their results thoroughly. In the user study, a couple of users mentioned that the given spreadsheets were easy, and they gave their answers quickly without further testing. A few of the answers turned out to be wrong or incomplete.

6.2 Recommendations

This section presents recommendations to tool builders in designing tool support for end users. Here is a list of general recommendations.

1. Integrate different evaluation techniques to validate the tool, and start the evaluation in the early design stage. In this thesis work, we informally interviewed a few spreadsheet users once the toolset prototype was available to gather their opinions and potential issues. Guideline-based evaluation was used before the user study to partly evaluate the usability quickly and inexpensively. The user experiments were valuable to further evaluate the toolset's usability, as well as its effectiveness. Finally, perceptual theories were applied to validate and explain the results from the guideline-based evaluation and user study.

2. Tool builders should use terms that end users are familiar with, rather than terms that are familiar to the tool designers. For example, in the SHriMP Query View, spreadsheet users had difficulties understanding terms such as "Restrict Node Type", "Incoming/Outgoing Arcs", and "Restrict Levels" unexpectedly, although these terms are familiar to computer scientists.

3. Tools should try to adopt notations that are commonly accepted by the majority of users, even though these notations might not always convey the best semantics. For example, one SHriMP-Cell user still chose the wrong arc direction even after this was explained over again in the user study, probably because the arc directions in SHriMP-Cell are opposite to the arrow directions in Excel. SHriMP-Cell arc directions show the dependency direction, while Excel arrow directions show how the data flows. It would be easier for spreadsheet users to learn SHriMP-Cell if arc directions were the same as Excel arrow directions, since end-users are already familiar with the Excel notation.

4. Working on spreadsheets, especially large ones, is very error-prone. Engineering principles can be applied. For example, pair programming techniques might be applied for spreadsheet users to reduce errors and improve quality [15]. During the user experiments, we observed that spreadsheet users often have to switch between different views, trace dependencies in various parts of the spreadsheet, and memorize or write down intermediate results. All these tasks could be eased and made less error-prone by two people working together, or with better tool support.

Based on our experience designing and evaluating SHriMP-Cell, we recommend that the following features in SHriMP be improved.

1. Provide an option to lay out nodes by column (or row). It would be nice if the SHriMP view can keep the column (or row) structure of a spreadsheet, because cells in the same column (or row) may follow the same pattern. An example SHriMP view of the "Income & Balance" sheet is shown in Figure 6.1 (where the layout was adjusted manually based on the cells' columns). This layout could help organize the nodes to reduce arc crossings, and be easier to map to the spreadsheet tabular layout.

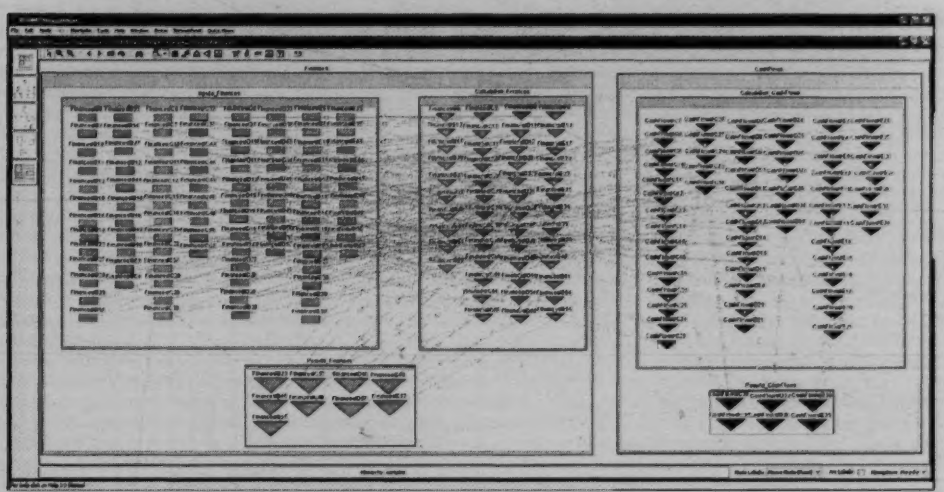


Figure 6.1: An example SHriMP view with column-based layout

2. Make the graph in the Query View interactive. The Query View tool is a powerful tool to detect indirect dependencies. It would be nice if users can check a node's attributes in the presented graph. Otherwise, users have to memorize the node name, and switch back to the main view to the check the node's attributes.

3. In the overview window, users should be able to click on anywhere inside the small red rectangle to move the overview focus, not just the edge, to ease interaction.

4. We propose a "selective arcs mode" to ease graph exploration when there are too many arc crossings. In this mode, all arcs are hidden by default. Arcs that are associated to a node will be shown when the user mouses over the node, and all direct and indirect arcs will be shown when a node is selected.

5. Add more user assistance or wizards for complex tasks. For example, both SHriMP-Cell users struggled with choosing the Query View options. It would be helpful if there were some descriptions about those options using spreadsheet notions.

6. Extend the search tool to support searching attributes. One user asked us whether he could search a cell node based on its label. This requires searching by attribute which is currently not supported in SHriMP.

6.3 Summary

This thesis proposes a reverse engineering method to extract artifacts from spreadsheet documents, build various abstractions, and represent spreadsheets in graphical views. A spreadsheet analyzer has been built and integrated with SHriMP and Excel to visualize spreadsheets, analyze cell dependencies, and detect and highlight anomalies. Both guideline-based and theory-based evaluation techniques have been applied to evaluate the usability of the toolset. Finally, a user study was conducted to further validate the usability and test the effectiveness of the dependency analysis features of the toolset by comparing it with Excel 2003's formula evaluator feature. The results show that our reverse engineering method is applicable to help end users better understand spreadsheets. The tool, SHriMP-Cell, is easy to use in general, and very effective in finding cell relationships, especially in detecting direct and indirect cell dependencies that cross different worksheets or referred cells that are beyond the boundaries of the screen.

6.4 Future work

We propose the following future work.

- Run a larger user study, and give users spreadsheet auditing or reuse tasks such as finding errors or extending a spreadsheet. Also compare how different abstractions help users to do certain tasks.

- Develop more clustering heuristics by studying more real-world spreadsheets. Add user assistance and layout information criteria such as spatial distance threshold and formatting rules to cluster cells more effectively.

- Try other visualization techniques, and investigate better techniques to integrate the spreadsheet analyzer with the spreadsheet application and the visualizer seamlessly.

For example, when a spreadsheet is changed, the visualization could be updated automatically without further user interaction.

- Once POI's "shared formula" defect is fixed, larger spreadsheets can be analyzed automatically to validate the tool's robustness and scalability.
- Find effective ways to detect and visualize the difference between different versions of spreadsheets so that things that are added, deleted, or changed can be easily highlighted.

Bibliography

[1] Apache POI Framework. http://jakarta.apache.org/poi/hssf/.

[2] Extensible Markup Language. http://www.w3.org/XML/.

[3] GNOME Gnumeric. http://www.gnome.org/projects/gnumeric/.

[4] J-Integra for COM. http://j-integra.intrinsyc.com/products/com/.

[5] Microsoft Visual J#. http://msdn.microsoft.com/vjsharp/default.aspx.

[6] OpenOffice Calc. http://www.openoffice.org/product/calc.html.

[7] Rigi home page. http://www.rigi.csc.uvic.ca/.

[8] SHriMP (Simple Hierarchical Multi-Perspective) website. http://www.thechiselgroup.org/shrimp/.

[9] The JACOB Project: A Java-COM Bridge. http://danadler.com/jacob/.

[10] Wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Information_visualization.

[11] XML Metadata Interchange. http://www.omg.org/technology/documents/formal/xmi.htm.

[12] R. Abraham and M. Erwig. Header and unit inference for spreadsheets through spatial analyses. In *VLHCC '04: Proceedings of the 2004 IEEE Symposium on Visual Languages - Human Centric Computing (VLHCC'04)*, pages 165–172, Washington, DC, USA, 2004. IEEE Computer Society.

[13] Adobe. Adobe Photoshop website. http://www.adobe.com/products/photoshop/.

[14] D. Ballinger, R. Biddle, and J. Noble. Spreadsheet visualization to improve end-user understanding. In *CRPITS '24: Proceedings of the Australian Symposium on Information Visualisation*, pages 99–109, Darlinghurst, Australia, Australia, 2003. Australian Computer Society, Inc.

[15] K. Beck. *Extreme Programming Explained: Embrace Change*. Addison-Wesley, 1999.

[16] J. G. Benjafield. *Cognition*. Prentice Hall, 1992.

[17] A. Blackwell. Cognitive Dimensions of Notations Resource Sites. http://128.232.0.20/ afb21/CognitiveDimensions/index.html.

[18] A. F. Blackwell, M. M. Burnett, and S. P. Jones. Champagne prototyping: A research technique for early evaluation of complex end-user programming systems. In *VLHCC '04: Proceedings of the 2004 IEEE Symposium on Visual Languages - Human Centric Computing (VLHCC'04)*, pages 47–54, Washington, DC, USA, 2004. IEEE Computer Society.

[19] S. C. Bloch. *Spreadsheet Analysis for Engineers and Scientists*. Wiley-Interscience, 1st edition, 1995.

[20] S. C. Bloch. *Excel for Engineers and Scientists*. Wiley, 2th edition, 2002.

[21] B. W. Boehm, B. Clark, E. Horowitz, J. C. Westland, R. J. Madachy, and R. W. Selby. Cost models for future software life cycle processes: COCOMO 2.0. *Annals of Software Engineering*, 1:57–94, 1995.

[22] K. R. Boff, L. Kaufman, and J. P. Thomas. *Handbook of Perception and Human Performance. Volume II. Cognitive Processes and Performance*. Wiley-Interscience Publication, 1986.

[23] D. Bricklin and B. Frankston. VisiCalc. http://www.bricklin.com/visicalc.htm.

[24] M. Clermont. Finding high-level structures in spreadsheet programs. In *WCRE '02: Proceedings of the 9th Working Conference on Reverse Engineering*, pages 221–232, Richmond, VA, USA, 2002. IEEE Computer Society.

[25] M. Clermont. Analyzing large spreadsheet programs. In *WCRE '03: Proceedings of the 10th Working Conference on Reverse Engineering*, pages 306–315, Washington, DC, USA, 2003. IEEE Computer Society.

[26] M. Clermont. *A Scalable Approach to Spreadsheet Visualization*. PhD thesis, 2003.

[27] M. Clermont, C. Hanin, and R. Mittermeir. A spreadsheet auditing tool evaluated in an industrial context. In *EuSpRIG 2002 symposium*, Cardiff, Wales, 2002.

[28] J. H. Cross II, E. J. Chikofsky, and C. H. May Jr. Reverse engineering. *Advances in Computers*, 35:199–254, 1992.

[29] M. Erwig and M. M. Burnett. Adding apples and oranges. *4th Int. Symp. on Practical Aspects of Declarative Languages*, pages 173–191, 2002.

[30] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[31] J. J. Gibson. *The Ecological Approach to Visual Perception*. Boston: Houghton Mifflin, 1979.

[32] B. E. Goldstein. *Sensation and Perception*. Wadsworth-Thomson Learning, 6th edition, 2002.

[33] T. Green and A. Blackwell. Cognitive Dimensions of Information Artefacts: a tutorial. http://www.cl.cam.ac.uk/ afb21/CognitiveDimensions/CDtutorial.pdf.

[34] J. Han and M. Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, 2001.

[35] C. D. Hansen and C. Johnson. *The Visualization Handbook*. Academic Press, 2005.

[36] D. Harrison and J. W. Yu. *The Spreadsheet Style Manual*. McGraw-Hill Education, 1990.

[37] D. G. Hendry and T. R. G. Green. Cogmap: a visual description language for spreadsheets. *Journal of Visual Languages and Computing*, 4(1):35–54, 1993.

[38] D. G. Hendry and T. R. G. Green. Creating, comprehending and explaining spreadsheets: a cognitive interpretation of what discretionary users think of the spreadsheet model. *International Journal of Human-Computer Studies*, 40(6):1033–1065, 1994.

[39] R. C. Holt, A. Schürr, S. E. Sim, and A. Winter. GXL home page. http://www.gupro.de/GXL/.

[40] IBM. IBM Lotus 1-2-3. http://www-142.ibm.com/software/sw-lotus/products/product2.nsf/wdocs/123home.

[41] T. Igarashi, J. D. Mackinlay, B.-W. Chang, and P. T. Zellweger. Fluid visualization of spreadsheet structures. In *VL '98: Proceedings of the IEEE Symposium on Visual Languages*, page 118, Washington, DC, USA, 1998. IEEE Computer Society.

[42] S. P. Jones, A. Blackwell, and M. Burnett. A user-centred approach to functions in Excel. In *ICFP '03: Proceedings of the eighth ACM SIGPLAN international conference on Functional programming*, pages 165–176, New York, NY, USA, 2003. ACM Press.

[43] A. Khan. Java Excel API. http://www.andykhan.com/jexcelapi/index.html.

[44] R. Kimchi, M. Behrmann, and C. R. Olson. *Perceptual Organization in Vision: Behavioral and Neural Perspectives*. Lawrence Erlbaum, 2003.

[45] S. Lammers. *Programmers at Work: Interviews With 19 Programmers Who Shaped the Computer Industry*. Tempus Books, 1989.

[46] J. A. Lawrence and B. A. Pasternack. *Applied Management Science : Modeling, Spreadsheet Analysis, and Communication for Decision Making*. John Wiley & Sons, 2nd edition, 2002.

[47] D. Marr. *Vision: A Computational Investigation into the Human Representation and Processing of Visual Information*. W H Freeman, 1982.

[48] Microsoft. Microsoft Excel Newsgroups. news:microsoft.public.excel.

[49] Microsoft. Microsoft Office Excel. http://office.microsoft.com/en-us/FX010858001033.aspx.

[50] D. L. Moise and K. Wong. An industrial experience in reverse engineering. In *WCRE '03: Proceedings of the 10th Working Conference on Reverse Engineering*, pages 275–284, Washington, DC, USA, 2003. IEEE Computer Society.

[51] D. L. Moise, K. Wong, and D. Sun. Integrating a reverse engineering tool with Microsoft Visual Studio .NET. In *CSMR '04: Proceedings of the Eighth European Conference on Software Maintenance and Reengineering (CSMR'04)*, pages 85–92. IEEE Computer Society, 2004.

[52] R. Molich and J. Nielsen. Improving a human-computer dialogue. *Commun. ACM*, 33(3):338–348, 1990.

[53] P. Moore and C. Fitz. Gestalt theory and instructional design. *Journal of Technical Writing and Communication*, 23(2):137–157, 1993.

[54] H. A. Müller, J. H. Jahnke, D. B. Smith, M.-A. Storey, S. R. Tilley, and K. Wong. Reverse engineering: a roadmap. *22nd International Conference on the Future of Software Engineering*, pages 49–60, 2000.

[55] H. A. Müller, S. R. Tilley, and K. Wong. Understanding software systems using reverse engineering technology perspectives from the Rigi project. In *CASCON '93: Proceedings of the 1993 conference of the Centre for Advanced Studies on Collaborative research*, pages 217–226. IBM Press, 1993.

[56] J. M. Nevison. *The Element of Spreadsheet Style*. Prentice Hall, 1987.

[57] J. Nielsen. Enhancing the explanatory power of usability heuristics. In *CHI '94: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 152–158, New York, NY, USA, 1994. ACM Press.

[58] J. Nielsen and R. L. Mack. *Usability Inspection Methods*. John Wiley & Sons, 1994.

[59] J. Nielsen and R. Molich. Heuristic evaluation of user interfaces. In *CHI '90: Proceedings of the SIGCHI conference on human factors in computing systems*, pages 249–256, New York, NY, USA, 1990. ACM Press.

[60] R. R. Panko. What we know about spreadsheet errors. *Journal of End User Computing (Special issue on Scaling Up End User Development)*, 10(2):15–21, 1998.

[61] R. R. Panko and R. P. H. Jr. Spreadsheets on trial: A survey of research on spreadsheet risks. *Proceedings of the Twenty-Ninth Hawaii International Conference on System Sciences*, January 1996.

[62] M. Petre, A. Blackwell, and T. Green. Cognitive questions in software visualisation. *Invited chapter in: Stasko, J., Domingue, J., Brown, M., and Price, B. (Eds.), Software Visualization: Programming as a Multimedia Experience. MIT Press*, pages 453–480, 1998.

[63] J. Preece, Y. Rogers, and H. Sharp. *Interaction Design: Beyond Human-Computer Interaction*. John Wiley & Sons, 2002.

[64] D. Rafiei, D. Moise, and D. Sun. Finding syntactic similarities between XML documents. In *Proceedings of the 1st International Workshop on XML Data Management Tools and Techniques (XANTEC'06)*, 2006.

[65] K. Rajalingham, D. R. Chadwick, and B. Knight. Classification of spreadsheet errors. *European Spreadsheet Risks Interest Group (EuSpRIG)*, 2001.

[66] D. Rentz. Microsoft Excel File Format. http://sc.openoffice.org/excelfileformat.pdf.

[67] M. B. Rosson and J. M. Carroll. *Usability Engineering: Scenario-Based Development of Human-Computer Interaction*. Morgan Kaufmann, 2002.

[68] J. Sajaniemi. Spreadsheet Visualization - The S2 and S3 Visualizations. http://www.cs.joensuu.fi/pages/saja/ui/ss-visual.htm.

[69] J. Sajaniemi. Modeling spreadsheet audit: A rigorous approach to automatic visualization. *Journal of Visual Languages and Computing*, 11:49–82, 2000.

[70] H. R. Schiffman. *Sensation and Perception: An Integrated Approach*. John Wiley & Sons, 5th edition, 2001.

[71] C. Sengupta. *Financial Modeling Using Excel and VBA*. Wiley, 2004.

[72] M.-A. Storey. Theories, tools and research methods in program comprehension: past, present and future. *Software Quality Control*, 14(3):187–208, 2006.

[73] M.-A. D. Storey, F. D. Fracchia, and H. A. Müller. Cognitive design elements to support the construction of a mental model during software exploration. *Journal of Systems and Software*, 44(3):171–185, 1999.

[74] M.-A. D. Storey, K. Wong, F. D. Fracchia, and H. A. Müller. On integrating visualization techniques for effective software exploration. In *Proceedings of IEEE Symposium on Information Visualization (InfoVis'97)*, 1997.

[75] M.-A. D. Storey, K. Wong, and H. A. Müller. How do program understanding tools affect how programmers understand programs? *Science of Computer Programming*, 36(2-3):183–207, 2000.

[76] D. Sun and K. Wong. On understanding software tool adoption using perceptual theories. In *ACSE 2004: Proceedings of the Fourth International Workshop on Adoption-Centric Software Engineering*, pages 51–55. Institution of Electrical Engineers, 2004.

[77] D. Sun and K. Wong. On evaluating the layout of UML class diagrams for program comprehension. In *IWPC 2005: Proceedings of the 13th IEEE International Workshop on Program Comprehension*, pages 317–326. IEEE Computer Society Press, 2005.

[78] Sun Microsystems. Java Beans. http://java.sun.com/products/javabeans/.

[79] S. Tilley and D. Smith. Coming attractions in program understanding. Technical Report CMU/SEI-96-TR-019, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 1996.

[80] M. Tukiainen. Evaluation of the cognitive dimensions questionnaire and some thoughts about the cognitive dimensions of spreadsheet calculation. In *PPIG-13: Proceedings of the 13th Annual Workshop of the Psychology of Programming Interest Group*, pages 291–301, 2001.

[81] University of Alberta - Faculty of Education. EDIT 202: Technology Tools for Teaching and Learning. http://www.quasar.ualberta.ca/edit202/.

[82] C. Ware. *Information Visualization: Perception for Design*. Morgan Kaufmann Publishers, 2000.

[83] K. Wong. *The reverse engineering notebook*. PhD thesis, 2000. Adviser - Halisi A. Müller and Adviser - Frank Ruskey.

[84] K. Wong and D. Sun. On evaluating the layout of UML diagrams for program comprehension. *Software Quality Journal*, 14(3):233–259, 2006.

[85] K. Wong, S. R. Tilley, H. A. Müller, and M.-A. D. Storey. Structural redocumentation: A case study. *IEEE Software*, 12(1):46–54, 1995.

Appendix A

Solicitation Letter

Dear xxx (fill in name of potential participant),

You are being invited to participate in a study entitled "evaluation of a spreadsheet visualization tool". The purpose of this study is to evaluate the usability and effectiveness of a spreadsheet visualization tool developed to ease end users understanding spreadsheets. We are interested in your participation because you have experience working with spreadsheets.

The procedure of this study will be the following: at the beginning of this study, we will introduce background knowledge related to the tool such as spreadsheet structures, visualization, etc. We will demonstrate the features that the tool provides and teach you how to use these features for some sample tasks. You will try using the tool and go over its functions. Then, you will be given some tasks related to understanding a spreadsheet, and asked to complete these tasks using the tool. After these tasks are finished, there will be a post-study questionnaire and an informal conversation to discuss the usability and effectiveness of the tool, your concerns, suggestions and issues you had using the tool. The whole study will take about 90 minutes. Your activities will be videotaped for further analysis. If you can complete the entire study, you will receive a gift certificate afterwards as a reward.

Your participation in the study is completely voluntary. You can withdraw from the study at any time, without explanation. You can take breaks and ask questions at any time during this study. Any data collected in the study will remain confidential and be kept on secure machines. Any analysis results that would be published will remain anonymous (that is, not identifiable to you).

If you are interested in participating in this study or have further questions, please contact me by email: dabo@cs.ualberta.ca or by phone (780) 492-9438.

Thank you and I look forward to hearing from you.

Regards,

Dabo Sun

M.Sc. Student

Department of Computing Science

University of Alberta

Appendix B

Pre-study questionnaire

Please note: you have the rights to refuse to answer any of these questions.

1. What is your background? e.g. your field of study, your profession, etc.
2. How long have you been using spreadsheets? How intensive have you been working on spreadsheets?
3. What spreadsheet applications have you used, e.g. Microsoft Excel, StarOffice Calc, etc.?
4. How many spreadsheets have you roughly made or worked on? What was the size (in cells) of the largest spreadsheet that involved calculation (i.e., it has formulas) you have done?
5. What do you mainly use spreadsheets for?
6. Have you had to understand other peoples spreadsheets? If so, did you feel it is hard to understand those spreadsheets created by other people?

Appendix C

Consent form

C.1 Purpose of the study

The purpose of this research study is to explore how we can design and build a tool to facilitate end users better understand spreadsheets. This study is important because spreadsheets are often error prone, hard to understand, maintain and reuse. As part of this study, you are invited to participate in evaluating the usability and effectiveness of the tool. Anonymized results will form part of a thesis, and might be published in computer science and software engineering journals.

C.2 Procedure

At the beginning of this study, we will introduce background knowledge related to the tool such as spreadsheet structures, visualization, etc. We will demonstrate the features that the tool provides and teach you how to use these features for some sample tasks. You can try using the tool and go over its functions. Then, you will be given some tasks related to understanding a spreadsheet, and you will be asked to complete these tasks by using the tool. After these tasks are finished, there will be an informal conversation discussing the usability and effectiveness of the tool, your concerns, suggestions and issues you had using the tool. The whole study will take about 90 minutes. Your activities will be videotaped for further analysis. You will receive a 25 CAD gift certificate afterwards if you can finish the entire study.

C.3 Your rights

- Your participation in the study is completely voluntary.

- You can withdraw from the study at any time, without explanation. Please note: you will not receive the gift certificate if you withdraw before you complete the entire study.
- You can take breaks and ask questions at any time.
- Any data collected in the study will remain confidential and be kept for five years on secure machines. At the end of this time, all data including questionnaires, videotapes and notes will be destroyed. If you choose to withdraw form this study, your data will be destroyed right away.
- Any analysis results that would be published will remain anonymous (that is, not identifiable to you).

C.4 Principal investigators

Dr. Kenny Wong and Dabo Sun
Department of Computing Science
University of Alberta Edmonton, Alberta,
Canada T6G 2E8

Please contact Dabo Sun by email dabo@cs.ualberta.ca or by phone (780) 492-9438 if you have further questions.

C.5 Agreement

Name of Participant:

As the Participant, I have been fully informed of the following statements before proceeding with the study:

- I understand the intent and purpose of this research, and my participation is completely voluntary.
- I understand that, upon my request, I have the right to withdraw from this study at any time and not have the data collected about me included in the analysis.
- I understand that during the course of the study, my activities will be videotaped.
- I understand that the experiment data will be kept for five years on secure machines. At the end of this time, all data including questionnaires, videotapes and notes will

be destroyed. If I choose to withdraw form this study, my data will be destroyed right away.

- I understand that my identity will be kept confidential, and my name will not be released in any publication about this research.
- I understand that after I finish the entire study, I will be given a 25 CAD gift certificate.

Participant's Signature:

Date:

Researcher's Signature:

Date:

Appendix D

Experimenter's handbook for SHriMP-Cell

D.1 General procedures

The purpose of this user study is to explore how end users understand spreadsheets, and evaluate tools to facilitate end-users in better understanding spreadsheets. The key of this study is to observe how users navigate and comprehend spreadsheets, to test whether the tools used in this study are easy to learn and use (good usability), as well as to examine what features of the tools are effective, and what could be improved. This is a controlled study in a laboratory environment. Therefore, the users should always be in control. However, experimenters should make the users feel as relaxed as possible and ease them not to feel any pressure to give the "right" answer.

D.1.1 Phases

All users are asked to fill out a pre-study questionnaire in order to collect and compare their experience in spreadsheets. Each experiment takes about 90 minutes and contains the following phases with approximate time limits in minutes indicated in brackets.

1. Orientation and consent form (5): Introducing this study and ask the user to sign the consent form.
2. Training (15): Going over the features that the tool provides to navigate and understand a spreadsheet.
3. Practice tasks (15): Some small tasks to help the user get familiar with the tool.
4. Formal tasks (40): Formal tasks to test the tool.

5. Post-study questionnaire (5): Evaluating the user's satisfaction and usability of the tool.

6. Post-study interview and finishing (10): Further discussing issues and debriefing.

The following table lists all the phases and time expected to finish each phase.

Tasks	Time (in minutes)
Orientation and consent form	5
Training	15
Practice	15
Formal tasks	40
Post-study questionnaire	5
Post-study interview and finishing	10
Total	**90**

D.1.2 Principles

The following principles should be followed during the experiments:

- **Quietness:** The lab should be quiet during the study. Disturbance should be minimized, especially during the formal task phase. Casual visitors should be avoided (post a sign on the door saying "user study in progress") and telephone calls need to be stopped.

- **Think aloud:** Always encourage the user to talk and let the experimenter know what he or she is thinking and doing ("think aloud").

- **Observation:** Experimenter should take as much notes on this handbook as possible. Write down how long each task actually takes and issues raised if there are any.

- **Questions:** Experimenter should be as helpful as possible answering user's questions in terms of tool features and clarifications, but NOT directly to the answers, especially in the formal tasks session.

D.2 Orientation

Time: about 5 minutes

Greet the user and introduce the purpose of this study. Outline the phases of the experiment and how much time expected to complete each phase.

Stress the point: the purpose of this study is to evaluate tools, not people. Relax and enjoy. If you have any questions regarding the questions or tool features at any time, feel free to ask the experimenter.

Review the consent form and inform the users about their rights.

D.3 Training

Time: about 15 minutes

Introduce some background of spreadsheet and the features in Excel 2003 that might be helpful to understand a spreadsheet by going through some simple tasks.

D.3.1 Setup

Open the computational view for the "Macs.xls" spreadsheet in SHriMP-Cell, and open the "Macs.xls" in Excel 2003.

D.3.2 Background

Talk about formula cell references including direct references and indirect references, as well as absolute and relative references. Identify data cells, input cells and formulas (calculation cells). The following concepts should be gone through in detail:

- **Formulas**: starts with a "=" sign, usually contains operators, functions, and operands. A formula can refer to other cells by using cell references.
- **Data cells**: Non-empty and non-formula cells.
- **Input cells**: Special kinds of data cells which are used by other formulas, i.e. input cells are used for calculation.
- **Absolute and relative references**: Cell reference describes a cell's address, e.g., *A1* means cell in the first column and first row. Absolute reference is denoted with a dollar sign ($), e.g., *=$A$1+$B1+C$1*. If a cell reference has no dollar sign, we call it a relative reference (default in Excel).
- **Direct and indirect references**: if a cell (say *A1*) (could be an input cell or a formula) is referred by a formula (say *B1*), we call formula *B1* directly refer to cell *A1*. If *B1* is also referred by another formula (say *C1*), we call formula *C1* indirectly refer to cell *A1*. For example, in the Powerbook worksheet, *C3* directly refers to *D1*; *D3*

indirectly refer to *D1*, because *D3* (directly) refers to *C3*, and *C3* (directly) refers to *D1*.

D.3.3 Key features of SHriMP-Cell

SHriMP-Cell (SHriMP for short) provides a nested graph view of a spreadsheet with nodes and arcs. In SHriMP view, nodes and arcs use different shapes to represent different artifacts and their relationship, e.g. triangles for formulas, and rectangles for data cells. Different colors are used to represent different worksheets.

SHriMP has different views representing a spreadsheet. *SHriMP Computational View* will be used in this study. It visualizes all the worksheets in one graph, and only cells involve in computation show up in this view, i.e. non-referred data cells are filtered out, because they have no contribution to the calculation.

Blocks: A block in SHriMP is a cluster of block nodes or cell nodes. Nodes inside a block are called "descendants".

- **Input block**: input data cells, i.e. non-formula cells that are referred by other formulas
- **Calculation block**: formulas that are referred to by other formulas
- **Result block**: the final result cells, i.e., formulas that no other formulas refer to (or depend on).

IMPORTANT: draw a diagram to show how the three blocks work.
The following features need to learned in this study:

1. **Excel Formula view** (formula auditing mode): from Excel Menu "Tools" → "Formula Auditing" → "Formula Auditing Mode" (or by shortcut key Ctrl + ‘)
2. **Node & Arc**: Shape (different types: formulas are triangles; data cells and blocks are rectangles), color (different sheets: orange for the first sheet, and green for the second sheet, ...), and filtering (check/uncheck the checkbox in the Node/Arc Filter window.).
3. **Arc directions**:
   - **Incoming**: other cell nodes depend on this cell node, i.e. this cell is referred by other cells.

- **Outgoing**: this cell node depends on other cell nodes, i.e., this cell refers to other cells.

4. **Navigation**: nodes can be moved around, especially when nodes or arcs overlap. Multiple nodes can be selected by holding the Ctrl key.

5. **Collapse node**: to Collapse/Expand All Descendants, double click on a "Block" node.

6. **Show node attributes** (mouse middle click, or shortcut Key "X"), If a block node is selected, all nodes inside it will be laid out.

7. **Home icon** (or shortcut key "h").

8. **Search & select**: in the search window, input "Search Pattern", and click on the "Search" button. Select one or more of the results, and click on the "Select" button.

9. **Query View**: only interesting nodes and their "neighbors" (connected by incoming or outgoing nodes) will be shown. Select one or more nodes, then open the Query View. "Restrict Node Types", "Restrict Arc Types", "Neighbors", and "Arc Direction" options need to be explained by examples.

10. **Zooming feature**: zoom in (shortcut key "=/+"); zoom out (shortcut key "-").

11. **Thumbnail view**: you can move the rectangle around by clicking on the border and dragging. The rectangle can also be resized by dragging the small red box in the lower right corner.

D.4 Practice

Time: about 15 minutes

D.4.1 Setup

Open the computational view for the "Weather.xls" spreadsheet in SHriMP-Cell, and open "Weather.xls" in Excel 2003.

D.4.2 Practice tasks

Ask the user to do the following tasks and help them if needed:

1. Browse the "Weather" spreadsheet, and switch between the formula auditing mode and editing mode in Excel. Try how trace precedents and trace dependents work, as well as the arrow removal icons.

2. Identify data cells and formula cells.

3. Identify absolute cell references and relative cell references.

4. Which cells do Cell "E5" and "F5" refer to, i.e., which cells do "E5" and "F5" depend on?

5. Which cells use cell "D5" for calculation, i.e., which cells refer to "D5"?

D.5 Formal tasks

Time limit: 40 minutes (strict)

D.5.1 Setup

Open the "Historical-Income-Statement-and-Balance-Sheets.xls" spreadsheet in Excel 2003. Make sure no potential interruptions: put a sign on the door; turn off the phone and cell phones.

Note: This session is strictly 40 minutes. There are 5 questions. The user should not spend more than 15 minutes on each question.

D.5.2 Formal tasks

Give the user a copy of the formal task sheets page by page. Give user the next question sheet when he/she finishes the previous task or he/she has spent more than 15 minutes on the task.

Ask the users to try their best to finish these tasks, but do not feel any pressure. They are not expected to complete all the tasks perfectly. Record how much time the user spends on each question.

List of the formal tasks:

1. Browse the given spreadsheet and explain what it does, and how it is structured. In addition, explain how the calculation works.

2. Suppose you changed the value of Cell "Finances!B27", what other cells' value might also have been changed?

3. In the "CashFlows" worksheet, the column of year 1999 is blank. Do you think this is a mistake or something is missing or unfinished here? Why or why not?

4. What cells does Cell "CashFlows!E16" depend on (i.e., refer to) directly and indirectly? Note, to save time, if the cell is in the "CashFlows" sheet, you do not need to write the sheet name. If the cell is in the "Finances" sheet, you can just underscore the cell reference.

5. Suppose the results of the given spreadsheet is wrong, and you know that the problem is probably in the input cells, as the person who manually input those numbers is careless. Now, you will need to find all the input data cells. Please list the cell references and/or ranges. Hint: input cells are non-formula data cells that are referred by other formulas. Note: to save time, if the cell is in the "Finances" sheet, you do not need to write the sheet name. If the cell is in the "CashFlows" sheet, you can just underscore the cell reference.

D.6 Post-study questionnaire

Time: about 5 minutes

Give the questionnaire form to the user and encourage the user to complete the questions quickly; first impressions are fine.

D.7 Post-study interview and debriefing

Time: about 10 minutes

Interview should focus on the user's experience with the tool.

Thank the user for their time and effort!

Appendix E

Experimenter's handbook for Excel

E.1 General procedures

The purpose of this user study is to explore how end users understand spreadsheets, and evaluate tools to facilitate end-users in better understanding spreadsheets. The key of this study is to observe how users navigate and comprehend spreadsheets, to test whether the tools used in this study are easy to learn and use (good usability), as well as to examine what features of the tools are effective, and what could be improved. This is a controlled study in a laboratory environment. Therefore, the users should always be in control. However, experimenters should make the users feel as relaxed as possible and ease them not to feel any pressure to give the "right" answer.

E.1.1 Phases

All users are asked to fill out a pre-study questionnaire in order to collect and compare their experience in spreadsheets. Each experiment takes about 90 minutes and contains the following phases with approximate time limits in minutes indicated in brackets.

1. Orientation and consent form (5): Introducing this study and ask the user to sign the consent form.
2. Training (15): Going over the features that the tool provides to navigate and understand a spreadsheet.
3. Practice tasks (15): Some small tasks to help the user get familiar with the tool.
4. Formal tasks (40): Formal tasks to test the tool.
5. Post-study questionnaire (5): Evaluating the user's satisfaction and usability of the tool.

6. Post-study interview and finishing (10): Further discussing issues and debriefing.

The following table lists all the phases and time expected to finish each phase.

Tasks	Time (in minutes)
Orientation and consent form	5
Training	15
Practice	15
Formal tasks	40
Post-study questionnaire	5
Post-study interview and finishing	10
Total	**90**

E.1.2 Principles

The following principles should be followed during the experiments:

- **Quietness:** The lab should be quiet during the study. Disturbance should be minimized, especially during the formal task phase. Casual visitors should be avoided (post a sign on the door saying "user study in progress") and telephone calls need to be stopped.
- **Think aloud:** Always encourage the user to talk and let the experimenter know what he or she is thinking and doing ("think aloud").
- **Observation:** Experimenter should take as much notes on this handbook as possible. Write down how long each task actually takes and issues raised if there are any.
- **Questions:** Experimenter should be as helpful as possible answering user's questions in terms of tool features and clarifications, but NOT directly to the answers, especially in the formal tasks session.

E.2 Orientation

Time: about 5 minutes

Greet the user and introduce the purpose of this study. Outline the phases of the experiment and how much time expected to complete each phase.

Stress the point: the purpose of this study is to evaluate tools, not people. Relax and enjoy. If you have any questions regarding the questions or tool features at any time, feel free to ask the experimenter.

Review the consent form and inform the users about their rights.

E.3 Training

Time: about 15 minutes

Introduce some background of spreadsheet and the features in Excel 2003 that might be helpful to understand a spreadsheet by going through some simple tasks.

E.3.1 Setup

Open the "Macs.xls" spreadsheet in Excel 2003.

E.3.2 Background

Talk about formula cell references including direct references and indirect references, as well as absolute and relative references. Identify data cells, input cells and formulas (calculation cells). The following concepts should be gone through in detail:

- **Formulas**: starts with a "=" sign, usually contains operators, functions, and operands. A formula can refer to other cells by using cell references.
- **Data cells**: Non-empty and non-formula cells.
- **Input cells**: Special kinds of data cells which are used by other formulas, i.e. input cells are used for calculation.
- **Absolute and relative references**: Cell reference describes a cell's address, e.g., *A1* means cell in the first column and first row. Absolute reference is denoted with a dollar sign ($), e.g., *=$A$1+$B1+C$1*. If a cell reference has no dollar sign, we call it a relative reference (default in Excel).
- **Direct and indirect references**: if a cell (say *A1*) (could be an input cell or a formula) is referred by a formula (say *B1*), we call formula *B1* directly refer to cell *A1*. If *B1* is also referred by another formula (say *C1*), we call formula *C1* indirectly refer to cell *A1*. For example, in the Powerbook worksheet, *C3* directly refers to *D1*; *D3* indirectly refer to *D1*, because *D3* (directly) refers to *C3*, and *C3* (directly) refers to *D1*.

E.3.3 Key features of Excel

1. **Formula view** (formula auditing mode): from Excel Menu "Tools" → "Formula Auditing" → "Formula Auditing Mode" (or by shortcut key Ctrl + ')

2. **Formula Evaluator**: introduce "Trace Precedents", "Trace Dependents", and "Remove All Arrows"

3. **Precedents**: precedents are cells that are used by (referred by) the current selected cell. Only formulas have precedents. Why? A cell's precedents could be input cells or formulas.

4. **Dependents**: dependents are cells that use (refer to) the current selected cell. Both input cells and formulas can have dependents, and dependents can only be formulas. Why?

IMPORTANT: Draw a diagram to explain trace precedents and dependents. Arrow directions shows how data "flows".

Ask the users whether there are any other features that they often use to understand a spreadsheet.

E.4 Practice

Time: about 15 minutes

E.4.1 Setup

Open the "Weather.xls" spreadsheet in Excel 2003

E.4.2 Practice tasks

Ask the user to do the following tasks and help them if needed:

1. Browse the "Weather" spreadsheet, and switch between the formula auditing mode and editing mode in Excel. Try how trace precedents and trace dependents work, as well as the arrow removal icons.

2. Identify data cells and formula cells.

3. Identify absolute cell references and relative cell references.

4. Which cells do Cell "E5" and "F5" refer to, i.e., which cells do "E5" and "F5" depend on?

5. Which cells use cell "D5" for calculation, i.e., which cells refer to "D5"?

E.5 Formal tasks

Time limit: 40 minutes (strict)

E.5.1 Setup

Open the "Historical-Income-Statement-and-Balance-Sheets.xls" spreadsheet in Excel 2003. Make sure no potential interruptions: put a sign on the door; turn off the phone and cell phones.

Note: This session is strictly 40 minutes. There are 5 questions. The user should not spend more than 15 minutes on each question.

E.5.2 Formal tasks

Give the user a copy of the formal task sheets page by page. Give user the next question sheet when he/she finishes the previous task or he/she has spent more than 15 minutes on the task.

Ask the users to try their best to finish these tasks, but do not feel any pressure. They are not expected to complete all the tasks perfectly. Record how much time the user spends on each question.

List of the formal tasks:

1. Browse the given spreadsheet and explain what it does, and how it is structured. In addition, explain how the calculation works.

2. Suppose you changed the value of Cell "Finances!B27", what other cells' value might also have been changed?

3. In the "CashFlows" worksheet, the column of year 1999 is blank. Do you think this is a mistake or something is missing or unfinished here? Why or why not?

4. What cells does Cell "CashFlows!E16" depend on (i.e., refer to) directly and indirectly? Note, to save time, if the cell is in the "CashFlows" sheet, you do not need to write the sheet name. If the cell is in the "Finances" sheet, you can just underscore the cell reference.

5. Suppose the results of the given spreadsheet is wrong, and you know that the problem is probably in the input cells, as the person who manually input those numbers was careless. Now, you will need to find all the input data cells. Please list the cell

references and/or ranges. Hint: input cells are non-formula data cells that are referred by other formulas. Note: to save time, if the cell is in the "Finances" sheet, you do not need to write the sheet name. If the cell is in the "CashFlows" sheet, you can just underscore the cell reference.

E.6 Post-study questionnaire

Time: about 5 minutes

Give the questionnaire form to the user and encourage the user to complete the questions quickly; first impressions are fine.

E.7 Post-study interview and debriefing

Time: about 10 minutes

Interview should focus on the user's experience with the tool.

Thank the user for their time and effort!

Appendix F

Post-study questionnaire

This post-study questionnaire gives you an opportunity to tell us your opinions of the usability of the tested tool. Please read each statement carefully and indicate how strongly you agree or disagree with the statement by circling a number on the scale.

1 - Strongly disagree

2 - Disagree

3 - Neutral

4 - Agree

5 - Strongly agree

If you have any additional comments, please write them down in the space provided to record them.

Questions

1. Overall, I was satisfied with how easy it was to use this tool.
2. The given spreadsheet was well organized in such a way that I was certain about the answers I gave.
3. I could effectively complete the given comprehension tasks using this tool.
4. Using this tool, I could effectively discover the dependencies between cells and formulas easily.
5. The interface of this tool was pleasant to use.
6. Using this tool, I was able to comprehend the structure of the given spreadsheets with ease.
7. This tool was simple to use.

8. I felt comfortable using this tool.

9. This tool was easy to learn.

10. This tool had all of the functions and capabilities that I expected it to have.

If you do not strongly agree with question 10, are there any features you wish it has, but not currently provided?

Other comments

Please write your comments here:

Appendix G

Post-study interview questions

1. After using the tool, what is your overall satisfaction? Are there any issues raised or any things could be improved?
2. Is the tool easy to learn in general? If there are some parts of the tools are hard to use, what are they?
3. Does the tool always keep you informed about what is going on, through appropriate feedback within reasonable time?
4. Is the tool good at discovering dependencies between cells and formulas?
5. Is the SHriMP view hard to map to the spreadsheet view? If so, what makes it hard? (for SHriMP-Cell)
6. For those tasks you have worked on, which ones can be easily done by using the SHriMP view, which ones can not? Why? (for SHriMP-Cell)
7. What features in the tool are most helpful in completing specific tasks? Users can refer the formal task sheets. (for Excel)
8. Do you have any suggestions to the procedure of this user study in general?